



Gobstones

Introducción a la Programación - Práctica 5

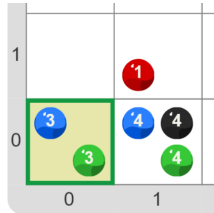
Alternativa condicional y funciones simples

CONSEJOS:

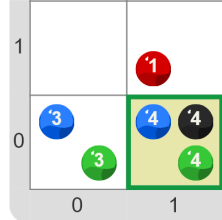
- Leer el enunciado en su totalidad y pensar en la forma de resolverlo **ANTES** de empezar a escribir código.
- Si un ejercicio no sale, se puede dejar para después y continuar con los ejercicios que siguen.
- Los ejercicios están pensados para ser hechos después de haber mirado la teórica correspondiente, y esta después de las actividades de indagación.
- Los ejercicios están tomados de las guías prácticas utilizadas en la materia de Introducción a la Programación de la Universidad Nacional de Quilmes por Pablo Ernesto "Fidel" Martínez López y su equipo, Introducción a la Programación de la Universidad Nacional de Hurlingham de Alan Rodas Bonjour y su equipo, de ejercicios y actividades realizadas por Federico Aloí y Miguel Miloro, a su vez basada en las guías Ejercicios de Introducción a la Programación del CIU General Belgrano, elaboradas por Carlos Lombardi y Alfredo Sanzo, y Fundamentos de la Programación del Proyecto Mumuki. Agradecemos a todos los que nos ayudaron con su inspiración.
- Realizar **EN PAPEL** los ejercicios que así lo indiquen.
- Si un ejercicio indica **BIBLIOTECA** significa que será útil para la realización de futuros ejercicios tanto en esta guía como en las siguientes, pudiendo ser utilizado sin tener que volver a definirlo. Es útil mantener registro de dichos procedimientos en su carpeta.

EJERCICIOS:

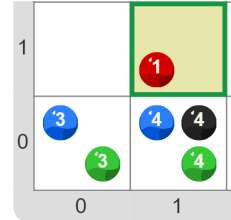
1. **EN PAPEL** Indicar el valor y el tipo que representan las siguientes expresiones en cada uno de los tableros A, B y C, suponiendo definido un procedimiento con el contrato dado al final.



(A)



(B)



(C)

- a. `not hayBolitas(Rojo)`
- b. `puedeMover(Sur) && puedeMover(Oeste)`
- c. `puedeMover(Sur) || puedeMover(Oeste)`
- d. `not puedeMover(Sur) && puedeMover(Oeste)`
- e. `nroBolitas(Negro) == nroBolitas(Azul)`
`&&`
`nroBolitas(Negro) == nroBolitas(Verde)`
- f. `puedeMover(opuesto(opuesto(dirección)))`
 suponiendo que, por alguna razón, esta expresión aparece dentro del cuerpo del procedimiento `Mover_SegúnColor_`, y que se lo invocó como
`Mover_SegúnColor_(Oeste, Azul)`
- g. ¿En qué situaciones al invocar `Mover_SegúnColor_` la siguiente expresión sería verdadera, en caso de aparecer dentro del cuerpo del procedimiento?
`not puedeMover(dirección)`
`&& not puedeMover(opuesto(dirección))`

El contrato del procedimiento dado es el siguiente:

```

procedure Mover_SegúnColor_(dirección, color)
  /* PROPÓSITO: Mover el cabezal en la dirección dada tantas
    celdas como el número de bolitas del color dado haya
    en la celda actual
  PRECONDICIONES:
    * hay al menos tantas celdas en la dirección dada como
    número de bolitas del color dado en la celda actual
  PARÁMETROS:
    * dirección: Dirección. La dirección hacia la cual
    mover el cabezal
    * color: Color. El color cuya cantidad de bolitas
    determina la cantidad a mover
  */
  
```

2. Definir funciones totales que sean verdaderas (describan al valor de verdad Verdadero) para cada uno de los siguientes casos. Recordar que es conveniente

utilizar funciones para expresar subtarear, de forma que las expresiones utilizadas no queden imposibles de entender. Recordar también que antes de escribir el código de una operación, debe escribirse el contrato de la misma (nombre, parámetros, propósito y precondiciones).

- a. Cuando la celda actual tiene más de 5 bolitas en total.
 - b. Cuando la celda actual tiene al menos 5 bolitas en total.
 - c. Cuando la celda actual tiene al menos 5 bolitas en total y el borde se encuentra justo al Este de la misma.
 - d. Cuando la celda actual tiene una celda lindante al **Norte** o al **Este**.
 - e. Cuando la celda actual tiene bolitas de todos los colores.
 - f. Cuando en la celda actual faltan bolitas de al menos un color (dar una solución sin usar la función del ítem anterior y otra usándola).
3. Escribir los siguientes procedimientos, recordando no mezclar niveles de abstracción del problema, para lo cual puede ser necesario definir otros procedimientos y/o funciones.
 - a. **SacarUnaFicha_SiSePuede(colorDeLaFicha)** que, dado el **colorDeLaFicha** que debe sacarse, saque una ficha siempre y cuando la misma esté en la celda. Si no hubiera fichas del color dado, el procedimiento no hace nada. Si hubiera varias fichas, solo debe sacar una.
OBSERVACIÓN: cada ficha se representa con una bolita del color correspondiente.
 - b. **DesempatarParaElLocal_Contra_(colorDelLocal,colorDelVisitante)** que, dados los colores de dos jugadores, ponga una bolita del **colorDelLocal** solamente en el caso en que la celda actual contiene la misma cantidad de bolitas de ambos colores.
 - c. **ExpandirBacteriaDeLaColonia()**, que siempre que en la celda actual haya un cultivo de bacterias y haya suficientes nutrientes, agregue exactamente una bacteria más y consuma nutrientes, a razón de dos nutrientes por bacteria expandida; si no hay bacterias o no hay suficientes nutrientes, no hace nada. Las bacterias se representan con bolitas Verdes y los nutrientes con bolitas Rojas.
 - d. **PonerFlecha_AlNorteSiCorresponde(colorDeLaFlecha)**, que dado un color para representar flechas, ponga una flecha al Norte si existe espacio para moverse en esa dirección. Las flechas al Norte serán representadas con una bolita del color dado.
4. Escribir los siguientes procedimientos:
 - a. **PudrirManzana()**, que en el caso de que en la celda actual haya al menos una manzana en buen estado y un gusano, pudre una manzana retirando un gusano.

OBSERVACIONES: las manzanas en buen estado se representan con bolitas de color Rojo, las manzanas podridas con bolitas de color Negro, y los gusanos con bolitas de color Verde.

SUGERENCIA: primero dar el contrato de la siguiente subtarea y utilizarlo en la resolución.

```
Reemplazar_Y_Por_(primerColorAReemplazar
                  , segundoColorAReemplazar
                  , colorAAgregar)
```

- b. Completar el código de `Reemplazar_Y_Por_`.
- c. `PudrirHasta_Manzanas(cantidadAPudrir)`, que pudre hasta un máximo de manzanas dada por la cantidad. Puede que se pudran menos manzanas si no se dan las condiciones necesarias (no hay suficientes manzanas en buen estado, o suficientes gusanos, por ejemplo).

5. **BIBLIOTECA** La combinación de parámetros y expresiones booleanas es interesante.

- a. Escribir un procedimiento `Poner_Si_(color, condición)` que dado un `color` y un valor de verdad llamado `condición`, ponga en la celda actual una bolita del `color` dado si el valor de verdad de la condición es verdadero, y no lo ponga si no.
EJEMPLO: `Poner_Si_(Rojo, nroBolitas(Rojo)==0)` solamente pone una bolita roja cuando no hay ninguna roja en la celda actual.
- b. Escribir los procedimientos `Sacar_Si_(color, condición)` y `Mover_Si_(dirección, condición)` que actúan de forma similar a `Poner_Si_`.
- c. Reescribir el procedimiento `DesempatarParaElLocal_` hecho antes, pero utilizando el procedimiento `Poner_Si_`.
- d. ¿Puede reescribirse el procedimiento `Reemplazar_Y_Por_` hecho antes, pero reutilizando únicamente los procedimientos `Poner_Si_` y `Sacar_Si_`? Si la respuesta es afirmativa, dar el código correspondiente. Si no, justificar por qué no sería posible.
- e. ¿Que beneficios trae tener los procedimientos `Sacar_Si_` y `Poner_Si_` contra utilizar `if` en cada caso?

6. El bosque, parte 4

En este ejercicio continuaremos expandiendo el dominio del bosque. Escribir los siguientes procedimientos. Considerar la reutilización de los procedimientos hechos en las partes anteriores y la definición de nuevas funciones necesarias para no tener que depender de la representación dada.

- a. **GerminarSemilla()**, que transforma una semilla en un árbol 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.
 - b. **AlimentarÁrboles()**, 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.
 - c. **ExplotarBomba()**, que explota una bomba en la celda actual, eliminando árboles. Al explotar, una bomba derriba 5 árboles en la celda actual y 3 en la celda lindante al Norte. Si la celda actual está en el borde Norte, entonces solo se eliminan los árboles de la celda actual. Atención que 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.
 - d. **Polinizar()**: los árboles en la celda actual polinizan la celda lindante en la dirección Este, generando tantas semillas en esa celda como árboles haya en la celda actual, menos 3. Por ejemplo, si en la celda actual hay 5 árboles, se generan 2 semillas en la celda lindante al Este. Si en la celda actual hay menos de 3 árboles, o no tiene lindante al Este, entonces no se hace nada.
7. **BIBLIOTECA** Escribir las siguientes funciones, para agregarlas a la biblioteca.
- a. **esCeldaVacía()**, que indica si la celda actual se encuentra vacía.
 - b. **hayAlMenosUnaDeCada()**, que indica si en la celda actual hay al menos una bolita de cada color.
 - c. **esCeldaConBolitas()**, que indica si la celda actual tiene al menos una bolita, de cualquier color.
8. **EN PAPEL** ¡Nuevamente Nova tiene problemas!
- a. Como Nova sigue confundido con las buenas prácticas de programación, nos consultó sobre cuál de las siguientes dos soluciones que se le ocurrieron es la correcta. El problema es sacar exactamente 8 bolitas de color **Azul** y la precondición, como es de esperar, es que haya al menos 8 bolitas azules en la celda actual. Explicarle a Nova cuál es la correcta, y por qué la otra no es una buena opción.

```

procedure SacarExactamente8BolitasAzulesOpciónA() {
  /* PROPÓSITO:
    * sacar exactamente 8 bolitas de color Azul
    PRECONDICIONES:
    * hay al menos 8 bolitas de color azul
  */
  repeat (8) { Sacar(Azul) }
}

procedure SacarExactamente8BolitasAzulesOpciónB() {

```

```

/* PROPÓSITO:
    * sacar exactamente 8 bolitas de color Azul
PRECONDICIONES:
    * hay al menos 8 bolitas de color azul
*/
repeat (8) { Sacar_Si_(Azul, hayBolitas(Azul)) }
}

```

- b. Ayudar a Nova a generalizar este procedimiento, escribiendo
SacarExactamente_Bolitas_(cantidadASacar, colorASacar)
9. Sobre el ejercicio “Soporte técnico” trabajado en la práctica anterior:
- a. Modificar la solución propuesta para agregar funciones donde resulte conveniente. ¿Qué ventajas se obtienen?
 - b. Modificar nuevamente la solución, teniendo en cuenta que ahora el procedimiento **RepararMáquina()**, ya NO debe tener como precondición que haya virus en la máquina actual, ya que podría haber tableros iniciales donde algunas máquinas no se hubieran infectado, y no debe pasarse un antivirus sobre una máquina que ya tiene la marca de Ok. Con este cambio, ¿el nombre del procedimiento debería seguir siendo el mismo?

10. ¡A la batalla!, parte 2

Escribir las siguientes funciones para el juego ¡A la batalla! de la práctica anterior, donde en las celdas del tablero se representan Soldados (los aliados con una bolita de color Negro y los enemigos con una bolita de color Rojo por cada soldado).

- a. **colorAliado()** y **colorEnemigo()** que describen el color de los aliados y los enemigos, respectivamente.
- b. **cantidadDeSoldadosDel_(colorDelEjército)**, que describe la cantidad de soldados de la celda actual del ejército dado.
- c. **esCeldaIndefensa()** que describe verdadero cuando no hay soldados aliados en la celda actual.
- d. **estadoDeEmergencia()** que describe verdadero solamente si existen más de 100 soldados enemigos, y además la celda está indefensa.
- e. **haySuficientesAliadosARazónDe_PorCada_(cantidadDefensa, cantidadAtaque)** que describe verdadero si hay por lo menos **cantidadDefensa** soldados aliados por cada **cantidadAtaque** soldados enemigos en la celda actual.

Pista: Piense en aplicar regla de tres simple donde:

cantidadDefensa de aliados --- cantidadAtaque de enemigos
 X cantidad de aliados --- Y cantidad en celda enemigos

- f. **aliadosNecesariosParaDefensaEficaz()** que describe el número de soldados aliados que faltan para defender la celda actual. Tener en cuenta que en la celda actual puede ser que haya soldados, pero que es

precondición de esta función que no hay suficientes aliados. Recordemos que 2 soldados enemigos pelean contra 3 soldados aliados y todos mueren.

11. **EN PAPEL** A continuación se dan una serie de funciones que se consideran primitivas, es decir, que puede asumir realizadas y no debe implementarlas de ninguna forma.

```
hayUnPlanetaA_Hacia_(distancia, dirección)
/*
    PROPÓSITO: Indica si hay un planeta a **distancia** celdas hacia
**dirección**.
    PARÁMETROS:
        * distancia: Número - La cantidad de celdas a la cual se
            desea buscar un planeta.
        * dirección: Dirección - La dirección hacia la cual mirar el planeta.
    PRECONDICIONES:
        * Hay al menos **distancia** celdas en dirección **dirección**.
        * El cabezal está sobre la nave.
    TIPO: Booleano
*/
```

```
combustibleRestante()
/*
    PROPÓSITO: Indica la cantidad de combustible que le queda a la nave.
    PRECONDICIONES:
        * El cabezal está sobre la nave.
    TIPO: Número
*/
```

Utilizando dichas funciones, se pide que se definan las siguientes, sin hacer suposiciones sobre la representación.

- a. **sePuedeAterrizarA_Hacia_(distanciaAPlaneta, direcciónAPlaneta)**, que asumiendo que el cabezal se encuentra sobre la nave y hay al menos **distanciaAPlaneta** celdas en dirección **direcciónAPlaneta**, indica si hay un planeta a **distanciaAPlaneta** en la dirección **direcciónAPlaneta** y sí el combustible es suficiente para llegar al mismo.
La nave consume una única unidad de combustible por cada celda que deba moverse.
- b. Sabiendo que el cabezal se encuentra sobre la nave y a exactamente 3 celdas de distancia de todos los bordes, se pide que escriba la función **hayUnPlanetaRecto()**, que indica que existe un planeta en cualquiera de las direcciones, a cualquier distancia desde la nave.

12. ¿Vamos al banco? - Parte 1

En este ejercicio utilizaremos el tablero de Gobstones para representar cuentas bancarias. Cada celda representará a una cuenta bancaria, y en cada una de ellas puede haber dinero en distintas monedas, que representaremos con distintos colores:

- bolitas negras para pesos argentinos.

- bolitas verdes para dólares estadounidenses.
- bolitas azules para euros.
- bolitas rojas para yuanes chinos.

Se pueden hacer tres operaciones: depósitos, extracciones y conversiones a divisa extranjera. Las extracciones pueden hacerse en cualquier moneda, pero los depósitos siempre serán en pesos.

En el caso en que se quiera depositar un monto en una moneda extranjera, se aplicará automáticamente la conversión a pesos según el precio de venta dado en la siguiente tabla:

Precios de venta	
1 dólar	80 pesos
1 euro	90 pesos
1 yuan	12 pesos


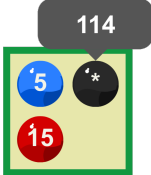

En cuanto a la conversión a divisa extranjera, el banco actualmente aplica las siguientes tarifas para la compra de divisa:

Precios de compra	
100 pesos	1 dólar
115 pesos	1 euro
17 pesos	1 yuan



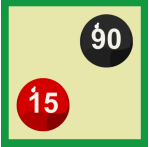
Realizar los siguientes procedimientos para poder manipular la cuenta:

- c. Depositar_EnMoneda_ComoPesos(cantidadADepositar, moneda),** que dada una cantidad de dinero a depositar y un color que representa la moneda en la que está representado ese monto, agrega a la cuenta la cantidad de pesos equivalente a lo indicado para depositar. En este procedimiento hay que aplicar la conversión indicada para el precio de venta.


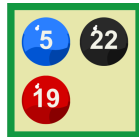

Ej.

		
Celda inicial	Depositar_EnMoneda_ComoPesos(2, Rojo)	Depositar_EnMoneda_ComoPesos(5, Verde)

- d. **ExtraerHasta_EnMoneda_(cantidadAExtraer, moneda)**, que dada una cantidad de dinero a extraer y un color que representa la moneda en la que se va a extraer, remueve de la cuenta la cantidad que se indica. Si no hubiera tanto dinero como el solicitado, se extrae todo lo que haya.

		
Celda inicial	ExtraerHasta_EnMoneda_(5, Rojo)	ExtraerHasta_EnMoneda_(10, Azul)

- e. **ConvertirHasta_PesosA_(pesosAConvertir, moneda)**, que dada una cantidad de pesos a convertir y un color que representa la moneda en la cual se quiere convertir, remueve los pesos de la cuenta y agrega la moneda solicitada. Si en la cuenta hubiera menos pesos de lo solicitado, se convierte todo lo que haya.

		
Celda inicial	ConvertirHasta_PesosA_(68, Rojo)	ConvertirHasta_PesosA_(100, Verde)

El último ejemplo es interesante: se piden convertir 100 pesos a dólares pero no hay 10 pesos en la cuenta, por lo que se va a intentar convertir el total de pesos que haya, 90. Con 90 pesos, no se llega a comprar ningún dólar, y como Gobstones solo trabaja con números enteros, no es posible tener medio dólar, por lo que queda en cero dólares.

- f. **RealizarCorridaCambiaria()**, que dado un tablero de 1 única fila y 10 columnas, donde cada celda representa una cuenta bancaria, se realiza una corrida cambiaria, donde en cada cuenta se cambia la totalidad de los pesos a dólares.

13. ¿Vamos al banco? - Parte 2

Continuaremos utilizando el mismo dominio del banco de la práctica anterior. Esta vez, vamos a realizar funciones que nos permitan abstraernos de la representación subyacente, así como simplificar cálculos en nuestras operaciones.

Se pide entonces que realice las siguientes funciones:

- a. **pesos()** que describe el color con el que se representan los pesos en el tablero, Negro.
- b. **dólares()** que describe el color con el que se representan los dólares en el tablero, Verde.
- c. **euros()** que describe el color con el que se representan los euros en el tablero, Azul.
- d. **yuanes()** que describe el color con el que se representan los yuanes en el tablero, Rojo.
- e. **ahorrosEn(moneda)** que dada una moneda, indica la cantidad de unidades de esa moneda en la cuenta actual.
- f. **cuantosDolaresSePuedeComprarCon_Pesos(cantidadDePesos)** que indica la cantidad de dólares que se pueden comprar con una cantidad de pesos dada.
- g. **cuantosEurosSePuedeComprarCon_Pesos(cantidadDePesos)** que indica la cantidad de euros que se pueden comprar con una cantidad de pesos dada.
- h. **cuantosYuanesSePuedeComprarCon_Pesos(cantidadDePesos)** que indica la cantidad de yuanes que se pueden comprar con una cantidad de pesos dada.
- i. **cuantosPesosSiVendo_Dólares(cantidadDeMonedaExtranjera)** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de dólares dada.
- j. **cuantosPesosSiVendo_Euros(cantidadDeMonedaExtranjera)** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de euros dada.
- k. **cuantosPesosSiVendo_Yuanes(cantidadDeMonedaExtranjera)** que indica la cantidad de pesos a obtener si se venden (depositan) la cantidad de yuanes dada.
- l. Vuelva a realizar los procedimientos de la práctica anterior, ahora utilizando las funciones realizadas en los puntos anteriores.

Reflexionamos: ¿Cuánto esfuerzo conlleva cambiar la representación de Euros y Pesos, para que ahora los primeros sean representados con bolitas negras y las segundas con azules.? ¿Cuántos lugares hubo que tocar? Sí la respuesta es más de 2, puede que no haya resuelto bien los ejercicios.