Algoritmos y Estructuras de Datos

Práctica 1

Instituto de Educación

UNIVERSIDAD NACIONAL DE HURLINGHAM

1. Programas y contratos

1) **Reemplazando bolitas:** Escribir un programa que reemplace una bolita de color roja con otra de color verde en la celda actual. Pruebe el programa en la computadora, modificando el tablero inicial de forma que el programa funcione satisfactoriamente.

2) **Moviendo bolitas:** Escribir un programa que mueva una bolita de color negro de la celda actual a la celda vecina al este, dejando el cabezal en la celda lindante al este.

3) **Poniendo en vecinas:** Escribir un programa que ponga una bolita de color azul en la celda vecina al norte de la actual.

4) **Analizando propósitos:** Dado el siguiente código

```
program {
    Poner(Verde)
    Sacar(Verde)
    Poner(Azul)
    Poner(Rojo)
  }
```

Diversos estudiantes realizaron propuestas para redactar su propósito, y también un profesor realizó explicaciones sobre cada una de estas propuesta. Asociar cada propuesta de propósito para el mismo (indicadas con las letras A, B, etc.) con la explicación que resulta correcta para dicha propuesta (indicadas con los números 1, 2, etc.)

A) Poner una bolita azul y luego una roja en la celda actual.	(1) No es un propósito, sino una descripción del funcionamiento. Para ser un propósito no debe preocuparse de los estados intermedios, solamente de la transformación final.
B) Agregar una bolita azul y una roja.	(2) Es un propósito incompleto ya que no establece los colores de las bolitas que se agregan. El propósito debe establecer con precisión la transformación esperada.
C) Pone una bolita verde y luego la saca, para a continuación poner una bolita azul y una roja.	(3) Es una enunciación correcta del propósito. El orden en que se agregan las bolitas es irrelevante, siempre que la celda actual finalice con una más de cada uno de los colores indicados.

D) Agregar una bolita roja y una bolita azul en la celda actual.	(4) Es un propósito incompleto, ya que no establece dónde se agregan las bolitas en cuestión. El propósito debe establecer con precisión la transformación esperada.
E) Agregar dos bolitas en la celda actual.	(5) Es una forma incorrecta de indicar la transformación esperada. Utiliza un lenguaje que sugiere un pensamiento operacional (o sea, centrado en las acciones individuales antes que en la transformación esperada.)

5) **Cuadrados verdes**. Escribir los siguientes programas:

   a) Uno que ponga un cuadrado de tamaño 3 con bolitas de color verde, con centro en la celda inicial (dejando el cabezal en dicha celda al finalizar).

      - ¿Qué ocurre si el tablero ya tenía bolitas verdes? ¿Es esto un problema, o el efecto obtenido es acorde al propósito?
      - ¿Qué ocurriría si desde la celda inicial no hay espacio para colocar las bolitas necesarias para que se cumpla el propósito?

   b) Uno que saque un cuadrado de tamaño 3 con bolitas de color verde (saca una bolita de cada celda), siendo la celda inicial el centro del cuadrado, dejando el cabezal en dicha celda al finalizar.

      - ¿Qué sucede si no hay al menos una bolita verde en cada una de las celdas necesarias? ¿Y si el tablero no tiene el tamaño adecuado?

6) **EN PAPEL**. Discutir en clase cuáles serían las precondiciones para programas cuyo propósito sea:

   a) Poner 1000 bolitas color Azul en la celda actual.

   b) Poner una bolita color Rojo en la celda lindante al Este de la celda actual y sacar una bolita Azul de la celda lindante al Oeste de la celda actual.

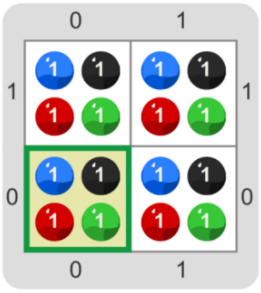
   c) Poner un rectángulo de bolitas Negras cuyo tamaño sea 3 filas y 5 columnas, centrado en la celda actual.

7) **Sacando un cuadrado**. Escriba un programa que saque del tablero un cuadrado multicolor de dos celdas de lado, donde la celda actual representa el vértice inferior izquierdo del mismo. Recuerde escribir primero el contrato del programa, y luego el código. Considere las siguientes preguntas como guía para escribir su programa:

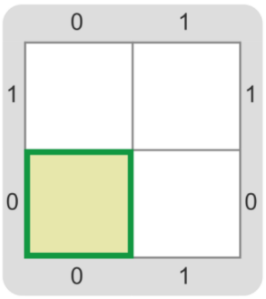
   a) ¿Que hace el programa? (Determina el propósito del programa)

   b) ¿Cuándo funciona tal cual se espera? (Determina la precondición del programa)

c) ¿Cómo lo hace? (Determina el código del programa)



Tablero inicial



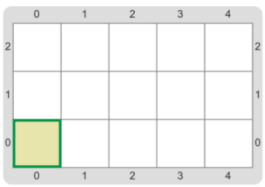
Tablero final

8) **Top-down**. Realizar cada uno de los siguientes puntos, en el orden dado (metodología top-down).

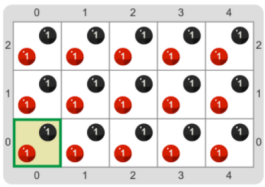
a) Escribir un procedimiento **DibujarRectánguloRojoYNegroDe5x3()** cuyo contrato es el siguiente:

```
procedure DibujarRectánguloRojoYNegroDe5x3()
      /*
     PROPÓSITO: Poner un rectángulo sólido de 5 celdas de ancho y 3
      celdas de alto, en la que cada celda tenga una bolita de color
      Rojo y una de color Negro. El cabezal comienza y termina en el
      vértice inferior izquierda del mismo.
      PRECONDICIONES: Hay al menos 4 celdas al
      Este y 2 celdas al Norte de la celda actual.
      /*
```

La metodología a seguir que se debe aplicar es la siguiente. En primer lugar, pensar una estrategia; se sugiere pensar una estrategia que involucre poner líneas de 5 celdas de ancho. Luego se debe definir primero el contrato de un procedimiento llamado **DibujarLíneaRojaYNegraDeTamaño5()** que exprese la subtarea sugerida. Para completar el ejercicio, se debe utilizar este procedimiento auxiliar en la codificación del procedimiento pedido, **DibujarRectánguloRojoYNegroDe5x3()**. El código del procedimiento auxiliar no es parte de este inciso, sino del siguiente.



Tablero inicial



Tablero final

b) Escribir el procedimiento **DibujarLíneaRojaYNegraDeTamaño5()** que quedó pendiente del ítem anterior. Para ello, se debe seguir la misma metodología que en el caso anterior: en primer lugar pensar una estrategia, luego

definir los contratos de los procedimientos que expresan las subtareas, y por último codificar el procedimiento pedido usando dichas subtareas (sin su código). Se sugiere que la estrategia involucre a subtareas llamadas **PonerUnaNegraYUnaRoja()** y **Mover4VecesAlOeste()**.

c) Escribir los procedimientos **PonerUnaNegraYUnaRoja()** y **Mover4VecesAlOeste()** que quedaron pendientes del ejercicio anterior. En este caso, los procedimientos se pueden expresar fácilmente utilizando comandos primitivos, por lo que no es necesario comenzar pensando en posibles subtareas.

9) **El bosque, parte 1**. En este ejercicio, se usará el tablero para representar un bosque. Cada celda representa a una parcela. Cada bolita verde representa un árbol. Cada bolita roja representa una semilla. Una bolita negra representa una bomba. Una bolita azul representa una unidad de nutrientes. Escribir los siguientes procedimientos de representación, que hacen lo que su nombre indica. Todos trabajan siempre sobre la celda actual.

- **PonerUnaSemilla()**
- **SacarUnÁrbol()**
- **PonerUnNutriente()**
- **SacarUnaSemilla()**
- **PonerUnaBomba()**
- **SacarUnNutriente()**
- **PonerUnÁrbol()**
- **SacarUnaBomba()**

2. Repeticiones y parámetros

1) **Dibujando un rectangulo con repeticiones**. Escribir un procedimiento **DibujarRectánguloRojoYNegroDe5x3()** que dibuje un rectángulo sólido de 5 celdas de largo por 3 de alto, símilar al realizado en 8)–a), pero esta vez, utilice repetición para solucionar el problema.

2) **Pintando el tablero** Escribir un procedimiento **PintarElTableroDeAzul()** que, asumiendo que el tablero tiene 10 celdas de largo y 7 celdas de alto, pinte absolutamente todo el tablero con bolitas azules, dejando exactamente una bolita azul en cada celda.

   a) ¿Cuál es la precondición del procedimiento?

   b) ¿Se le ocurre otra estrategia para resolver el problema?

   Importante Recuerde que la estrategia de solución debe quedar clara a partir de la lectura del código. Use subtareas con nombres apropiados para dicho objetivo.

3) **Moviendo tres veces a donde quieras**. Escribir un procedimiento **Mover3VecesAl_(direcciónAMover)** que dada una dirección direcciónAMover mueva el cabezal tres posiciones en dicha dirección.

   a) ¿Qué hay que hacer ahora para mover el cabezal tres veces al Norte?

   b) ¿Qué beneficios trae el uso de parámetros?

c) ¿Cuántos procedimientos puedo ahorrarme haciendo un único procedimiento con un parámetros?

¡Recordar! No olvidar escribir el contrato del procedimiento ANTES de realizar el código (y que los parámetros son parte del mismo); también discutir la precondición escrita con sus compañeros para verificar que la misma es adecuada y correcta.

4) **Y 6 de lo que quieras**. Escribir un procedimiento **Poner6DeColor_(colorAPoner)** que dado un color colorAPoner ponga 6 bolitas del color dado.

5) **Poner de a muchas**

BIBLIOTECA. Escribir un procedimiento **Poner_DeColor_(cantidadAPoner, colorAPoner)** que dado un número cantidadAPoner y un color colorAPoner, ponga tantas bolitas como se indica del color dado de la celda actual.

6) **Moviendo tantas veces como quieras a donde quieras**

BIBLIOTECA. Escribir **Mover_VecesAl_(cantidadAMover, direcciónAMover)**, un procedimiento que dado un número cantidadAMover y una dirección direcciónAMover mueva el cabezal tantas veces como la dada en dicha dirección.

7) **Sacar de a muchas**

BIBLIOTECA. Escribir un procedimiento **Sacar_DeColor_(cantidadASacar, colorASacar)** que dado un número cantidadASacar y un color colorASacar, saque tantas bolitas como se indica del color dado de la celda actual.

8) **Moviendo y poniendo**. Escribir un procedimiento **Poner_Al_(colorAPoner, direcciónDondePoner)** que dado un color colorAPoner y una dirección direcciónDondePoner, ponga una bolita del color dado en la celda vecina en la dirección dada, dejando el cabezal en dicha celda.

   a) ¿Cuántos casos distintos habría que considerar si no se usaran parámetros en este caso?

   b) ¿Cómo debería invocar al procedimiento para que ponga una bolita Azul en la celda al Norte?

   c) ¿Y sí quisiera una Azul y una Roja?

9) **Reemplazando colores**. Escribir **ReemplazarUnaDe_Por_(colorAReemplazar, colorPorElCualReemplazar)**, un procedimiento que dado un primer color colorAReemplazar y un segundo color colorPorElCualReemplazar, reemplaza una bolita del primer color por una del segundo color (En la celda actual).

10) **El bosque, parte 2**. Continuaremos representando el bosque que comenzamos en la práctica anterior. Esta vez queremos ser capaces de poner o sacar múltiples elementos de una sola vez.

    Importante: para realizar este ejercicio se espera haya realizado partes anteriores de este dominio, si aún no lo hizo, se recomienda volver y realizar el mismo previo a solucionar el ejercicio actual.

    - **Poner_Semillas(cantidadDeSemillasAPoner)**
    - **Poner_Árboles(cantidadDeÁrbolesAPoner)**
    - **Poner_Nutrientes(cantidadDeNutrientesAPoner)**
    - **Sacar_Semillas(cantidadDeSemillasASacar)**
    - **Sacar_Árboles(cantidadDeÁrbolesASacar)**
    - **Sacar_Nutrientes(cantidadDeNutrientesASacar)**

11) **¡A la batalla!, parte 1**

    Suponiendo que se está programando un juego 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), escribir los siguientes procedimientos:

    a) **EnviarAliadosParaDuplicarEnemigos()**, que agrega soldados aliados en la celda actual en cantidad suficiente para que haya el doble de aliados que de soldados enemigos.

    b) **PelearLaBatalla()**, que simula una batalla, suponiendo que hay suficiente cantidad de soldados aliados como para ganar la batalla. Durante una batalla, 2 soldados enemigos pelean contra 3 soldados aliados y todos mueren. Por ejemplo, si hay 6 enemigos y 10 aliados, mueren los 6 enemigos y 9 de los aliados; si hay 10 enemigos y 21 aliados, mueren los 10 enemigos y 15 soldados aliados.

       PISTA: ¿Qué cuenta hay que hacer para saber cuántos soldados aliados morirán?

12) **Sacando todas las de un color**

    BIBLIOTECA. Escribir un procedimiento **SacarTodasLasDeColor_(colorASacar)**, que quite de la celda actual todas las bolitas del color indicado por el parámetro.

    PISTA: Considerar utilizar el procedimiento **Sacar_DeColor_**, definido anteriormente. ¿Qué argumentos se le deberían pasar?

13) **¿Y si vaciamos la celda?**

    BIBLIOTECA. Escribir un procedimiento VaciarCelda() que quite de la celda actual todas las bolitas de todos los colores, dejando la celda vacía.

3. Alternativa condicional

1) **Sí se puede, sí se puede...**

   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 cuyos puntos se representan mediante la cantidad de bolitas del color del jugador, otorgue un punto al jugador con color olorDelLocal 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 serán representadas con una bolita del color dado.

2) **Hacer solo si...**

   La combinación de parámetros y expresiones booleanas es interesante.

   a) **BIBLIOTECA.** 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) == 2)** solamente pone una bolita roja cuando hay exactamente dos rojas en la celda actual.

   b) **BIBLIOTECA.** Escribir el procedimiento **Sacar_Si_(color, condición)** que actúa de forma similar al anterior, pero ahora sacando bolitas si la condición se cumple.

c) **BIBLIOTECA.** Escribir el procedimiento **Mover_Si_(dirección, condición)** que actúa de forma similar a los anteriores, pero ahora moviendo solo sí se cumple la condición dada.

d) ¿Que beneficios trae tener los procedimientos **Sacar_Si_** y **Poner_Si_** contra utilizar if en cada caso?

4. Funciones simples

1) **BIBLIOTECA.** Escribir las siguientes funciones, para agregarlas a la biblioteca.

   a) **esCeldaVacía()**, que indica si la celda actual se encuentra vacía.
   b) **tieneUnaDeCada()**, que indica si la celda actual tiene al menos una bolita de cada color.
   c) **esCeldaConBolitas()**, que indica si la celda actual tiene al menos una bolita, de cualquier color.

2) ¡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) **cantidadDeSoldadosDe_(colorDelEjército)**, que describe la cantidad de soldados de la celda actual del ejército dado.

   b) Vuelva a escribir **EnviarAliadosParaDuplicarEnemigos()**, y **PelearLaBatalla()**, que realizó en la versión anterior, ahora haciendo uso de la función hecha en el punto a, y luego conteste: ¿Realizó procedimientos auxiliares para resolver el problema? ¿Cuál de los códigos (entre este y el de la práctica anterior) comunica mejor la solución al problema? ¿Por qué?

   c) **esCeldaindefensa()** que indica si no hay soldados aliados en la celda actual.

   d) **estadoDeEmergencia()** que indica si existen más de 100 soldados enemigos, y además la celda está indefensa.

   e) **hayAlMenos_Aliados PorCada_Atacantes(cantidad Defensa, cantidadAtaque)** que indica si hay por lo menos cantidad Defensa soldados aliados por cada cantidadAtaque soldados enemigos en la celda actual. Por ejemplos si en la celda actual hubiera 10 soldados aliados y 5 enemigos, la función invocada como **hayAlMenos_AliadosPorCada_Atacantes(2, 1)**, describiría Verdadero, pues hay al menos dos aliados por cada atacante. Si se invocara con esos mismos argumentos en una celda donde hay 7 aliados y 5 enemigos, describiría Falso.

   f) **aliadosNecesariosParaDefensa EficazContra_(cantidadDeSoldados EnemigosAdicionales)** que describe el número de soldados aliados que faltan para defender la celda actual si a ella se suman la cantidad de soldados

enemigos dada. 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.

3) ¡Mira mami! ¡sin bolitas!

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**.
  PRECONDICIONES:
  - Hay al menos **distancia** celdas en dirección **dirección**.
  - El cabezal está sobre la nave.
  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.
  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 distanciaAPlanera 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.

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.

4) El bosque, parte 4.

Continuaremos utilizando el mismo dominio del bosque que venimos utilizando en las prácticas anteriores. Esta vez se pide escribir los siguientes procedimientos que modelan el bosque. 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) Escribir las funciones **árbol()**, **semilla()**, **bomba()**, **nutriente()** que describen las representaciones de los elementos del ejercicios «El bosque», de las prácticas anteriores.

b) **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.

c) **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.

d) **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: 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.

e) **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.

f) Debería haberle surgido la necesidad de contar con algunas funciones de representación adicionales, como **cantidad DeÁrboles()** o **cantidad De Nutrientes()** (entre otras). Sí el código de sus soluciones habla de bolitas en lugar del problema, recomendamos que revise nuevamente para intentar generar funciones que le permitan abstraerse de la representación.

5. Repetición Condicional

1) Un nuevo **IrAlBorde**.

Definir el procedimiento **IrAlBorde_(dirección)**, que lleva al cabezal al borde dado por el parámetro dirección. ¡Atención! Debe realizar el ejercicio sin utilizar el comando primitivo **IrAlBorde**. Dado que el único otro comando primitivo que permite mover el cabezal es Mover, debe repetirse su uso mientras nohasta que se

haya cumplido el propósito. Reflexionamos ¿Cuál es la condición que indica que el propósito se cumplió?

2) Otra forma de sacar todas

   Volver a definir el procedimiento Sacar **TodasLasDeColor_(color)**, que quita todas las bolitas del color dado por el parámetro color de la celda actual, pero esta vez SIN utilizar la expresión primitiva nroBolitas (directa o indirectamente).

5.1. Recorridos por celdas de la fila o columna

3) Vaciando una fila

   Considerar el procedimiento **VaciarFilaDe_(color)**, que debe quitar todas las bolitas del color dado por el parámetro **color** de cada una de las celdas de la fila actual. El cabezal puede empezar en cualquier celda de la fila, y también puede terminar en cualquier celda de la fila (ya sea celda inicial o cualquier otra).

   a) Definir el procedimiento, como siempre, comenzando por establecer el contrato, y luego recién el código.

   b) ¿La solución dada funciona si el cabezal se encuentra en medio de una fila? Si no es así, corregir el programa para que funcione en este caso también.

   c) Al recorrer la fila, ¿en qué dirección se movió el cabezal? ¿Podría haberse movido en la dirección opuesta?

   d) A partir de la respuesta anterior, ¿de cuántas formas posibles se puede realizar el recorrido de una fila?

   e) Volver a definir el procedimiento con direcciones distintas.

   f) ¿Y si tuviéramos que vaciar la columna en lugar de la fila? ¿Qué cambia entre las distintas formas de moverse?

4) Vaciando una fila hacia…

   Defina ahora el procedimiento **VaciarFilaDe_HaciaEl_(color, dirección)**, que debe quitar todas las bolitas del color dado por el parámetro **color** de cada una de las celdas de la fila actual, desde la celda en donde se encuentra el cabezal (incluyendo esta) hacia el final de la fila en la dirección dada por dirección. Tras definir el código y el contrato considere.

   a) ¿Qué valores puede tomar el parámetro dirección para que el propósito sea consistente con su nombre?
   b) ¿Qué nombre podría recibir el procedimiento para que sea correcto utilizarlo con cualquier dirección?

5.2. Recorridos por celdas del tablero

5) Vaciando un tablero

En cada uno de los casos siguientes, definir de la forma indicada el procedimiento **VaciarTableroDe_(color)**, que quite todas las bolitas del color dado por el parámetro color de cada una de las celdas del tablero. El cabezal debe poder comenzar en cualquier celda del tablero, y no es relevante para el problema donde finaliza, basta con que lo declare en su propósito.

Estructurar el procedimiento como un recorrido sobre las filas. ¿Qué subtareas van a precisarse en este caso? ¿Es necesario volver a definirlas o se pueden encontrar en esta práctica?

Estructurar el procedimiento como un recorrido sobre las celdas del tablero. Las subtareas necesarias serán diferentes, y puede ser que sea necesario definir alguna que aún no está disponible en esta práctica.

En este última opción, ¿Cuántas formas distintas hay de recorrer el las celdas del tablero? ¿Se podría elegir valores distintos para los movimientos? Implemente ahora nuevamente el procedimiento de 2 formas distintas.

Reflexionamos ¿Qué diferencias hay entre los recorridos anteriores del punto a y del b? ¿Qué subtareas son más complejas en cada caso? ¿Podrían considerarse otros recorridos que no fueran sobre celdas o filas?

6) Las subtareas más útiles de la historia

BIBLIOTECA. Escribir los procedimientos y las funciones necesarias para generalizar la noción de recorrido por celdas de un tablero, para que las direcciones de recorrido no estén fijas. En particular, definir (como siempre, comenzando por los contratos):

a) **IrAPrimeraCeldaEnUnRecorridoAI_Y_(dirPrincipal, dirSecundaria)**
b) **haySiguienteCeldaEnUnRecorridoAl_Y_(dirPrincipal, dirSecundaria)**
c) **IrASiguienteCeldaEnUnRecorridoAI_Y_(dirPrincipal, dirSecundaria)**

Que hacen precisamente lo que sugiere su nombre, permitiendo utilizarlas en un recorrido por celdas. Puede probarlas intentando colocar una bolita en cada celda del tablero, o volviendo a implementar el ejercicio anterior, ahora de forma parametrizada. Al escribir las precondiciones, tener en cuenta que las direcciones no pueden ser cualesquiera, sino que deben estar relacionadas. ¿Cuál es esa relación? ¿Cómo expresarla?

7) Y ahora más cosas sobre el tablero

   Escribir ahora los siguientes procedimientos, teniendo en cuenta que para todos, el cabezal puede comenzar en cualquier lugar del tablero, y terminar en dónde usted crea conveniente.

   a) **PintarTableroDe_(color)** que coloca exactamente una bolita del color dado en cada celda del tablero.

   b) **UnaDeCadaEnTodoElTablero()** que coloca una bolita de cada color en cada celda del tablero.

   c) **RellenarCon_EnAusenciaDe_EnElTablero(colorAPoner, colorAMirar)** que coloca una bolita de color colorAPoner en cada celda del tablero en la que no haya al menos una bolita de color colorAMirar.

   d) **CompletarHasta_De_EnElTablero (cantidad, color)** que deja en cada celda del tablero exactamente tantas bolitas del color dado como la cantidad indicada por el parámetro cantidad. Note que puede que ya existan bolitas del color dado en algunas de las celdas, en cuyo caso. Realice el procedimiento sin hacer uso del comando Sacar ni ninguno de los procedimientos que implican Sacar.

5.3. RECORRIDOS DE BÚSQUEDA

8) Buscando la bolita roja en la fila/columna

   Escribir un procedimiento **IrHastaLaBolita RojaHacia_(direcciónABuscar)** que deja el cabezal posicionado en la celda más próxima a la actual en la dirección dada que posea una bolita de color Rojo Cuidado, sí hay una bolita de color Rojo en la celda actual, el cabezal debe moverse a la más cercana, no permanecer en la actual. ¿Cuál es la precondición de este procedimiento?

9) Buscando la celda vacía

   Escribir un procedimiento **IrALaSiguienteVacíaHacia_(dirección)** que posiciona el cabezal en la próxima celda vacía en la fila o columna, desde la celda en donde se encuentra el cabezal (sin incluirla) hacia el borde en la dirección dada, dejando el cabezal en el borde en caso de no haber ninguna celda vacía en dicha dirección.

10) Buscando en todo el tablero

    Definir un procedimiento **IrHastaLaQueTengaUnaDeCada()** que posiciona el cabezal en cualquier celda que contenga una bolita de cada color, y que hace BOOM si no hubiera en el tablero alguna celda que cumpla con dicha característica.

11) EN PAPEL. Dadas las siguientes primitivas que modelan partes de un juego de ajedrez:

```
function hayUnaPiezaNegra()
  /*
  PROPÓSITO: Indica si hay una pieza negra en la celda actual.
  PRECONDICIONES: Ninguna
  */
```

```
procedure ComerPiezaNegra()
  /*
  PROPÓSITO: Come la pieza negra en la celda actual.
  PRECONDICIONES: Hay una pieza negra en la celda actual
  */
```

```
procedure MoverTorreBlancaHacia_(direcciónAMover)
  /*
  PROPÓSITO: Mueve la torre blanca una celda en la dirección dada.
  PARÁMETROS: direcciónAMover. Dirección - La dirección hacia la cual
  mover la pieza.
  PRECONDICIONES: Hay una celda en la dirección dada.
  */
```

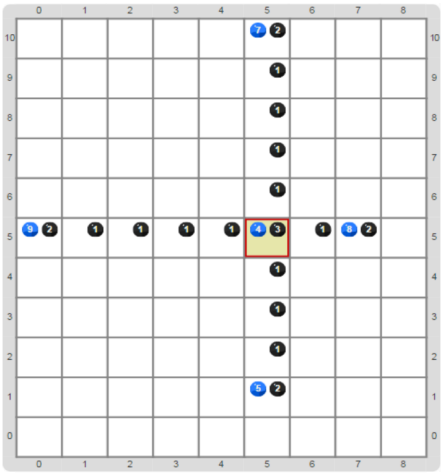
```
procedure ComerPiezaNegraConTorreHacia_(direcciónAComer)

  /*
  PROPÓSITO: Asumiendo que se está sobre una torre blanca, come la pieza
  negra más próxima a la celda actual hacia la dirección dada, dejando
  la torre en dicha celda.
  PARÁMETROS: direcciónAComer. Dirección - La dirección hacia la cual
  comer la pieza negra.
  PRECONDICIONES:
  - Se está ubicado en una celda con una torre blanca.
  - Existe al menos una pieza negra en alguna celda en la dirección
  dada desde la posición actual de la torre.
  */
```

Se pide que escriba el procedimiento **ComerPiezaNegraConTorreHacia_(direcciónAComer)** que asumiendo que se está sobre una torre blanca, come la pieza negra más próxima a la celda actual hacia la dirección dada, dejando la torre en dicha celda.

12) Se desea modelar el movimiento de mercadería en una sencilla red de depósitos, que tiene un depósito central, más un depósito local para cada punto cardinal. Para esto, se va a representar en el tablero un mapa muy simplificado.

- Tres bolitas negras marcan el depósito central,
- Dos bolitas negras marcan un depósito local,
- Una bolita negra marca el camino de central a local,
- Cada bolita azul marca una unidad de mercadería.



Los depósitos locales forman una cruz, donde el centro es el depósito central. No se sabe a qué distancia están los depósitos locales del depósito central.