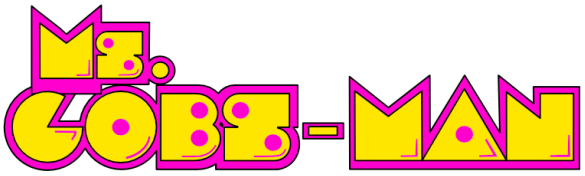


Práctica integradora de Funciones Simples y Con Procesamiento, Alternativa de Expresiones y Variables

ATENCIÓN: Se recomienda realizar esta práctica íntegramente en papel, y recurrir a la computadora solamente cuando así lo solicite el enunciado.

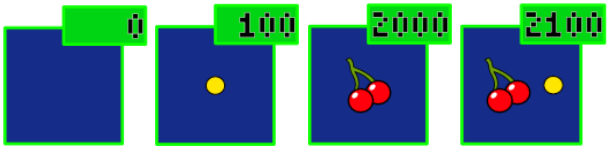
Ms. Gobs-Man es la secuela del popular juego de video argentino desarrollado en Gobstones, Gobs-Man. Ms. Gobs-Man incorpora una serie de características que se esperan transformen al juego en un éxito inmediato.

Esta vez el jugador se pondrá en la piel de Ms.Gobs-Man, la versión femenina de nuestro aclamado héroe comecocos. Como en la primer versión del juego, el objetivo será lograr que Ms. Gobs-Man se coma todos los cocos y las cerezas. Para ello contaremos con todas las primitivas que se presentaron en el primer juego, que ahora actúan sobre Ms.Gobs-Man, así como también la siguiente, que se agrega a las anteriores:

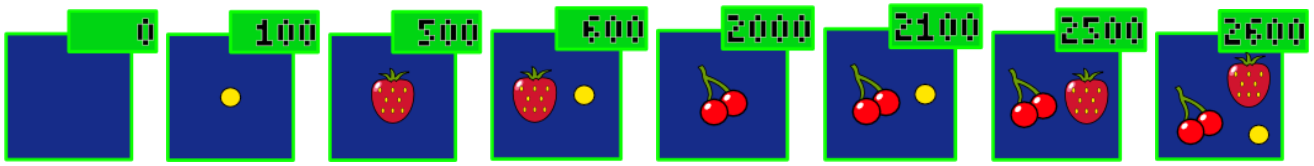
```
function hayCoco()
{- PROPÓSITO: Indica si hay un coco en la celda actual.
   PRECONDICIONES: Ninguna.
-}
```

Esta vez, sin embargo, tanto los cocos como las cerezas otorgarán puntaje al jugador.

Ejercicio 9)
Dado que el cabezal se encuentra en alguna celda, se espera poder determinar cuántos puntos obtendrá Ms. Gobs-Man si come todo lo que hay en dicha celda. Notar que la celda puede tener un coco, una cereza, ambos o estar vacía. Un coco otorga a Ms. Gobs-Man 100 puntos, y una cereza otorga 2000 puntos. Escribir la función: **`puntajeAObtenerEnCeldaActual`** que describe los puntos a obtener en la celda actual. A continuación se muestran algunas posibles celdas a analizar y los puntos que se deberían obtener:



Ejercicio 10)
Se plantea ahora que además de cerezas y cocos, las celdas pueden contener frutillas (O fresas, si prefiere). Las fresas otorgan 500 puntos solamente, pero pueden encontrarse en cualquier celda, por lo que ahora tenemos las siguientes posibilidades:



Replantear la función **puntajeAObtenerEnCeldaActual** para tener en cuenta dicha situación. Si la estrategia elegida anteriormente fue buena, entonces este cambio no debería redundar en demasiado trabajo. Si por el contrario la solución no fue buena, llevará más esfuerzo (Si fuera este último caso, repensar si se está separando el problema en las subtareas correctas, o ver si se puede mejorar). ¡Ah, por cierto! Casi se nos olvida, también se cuenta con la primitiva siguiente:

```
function hayFrutilla()
{- PROPÓSITO: Indica si hay una frutilla en la celda actual.
   PRECONDICIONES: Ninguna.
-}
```

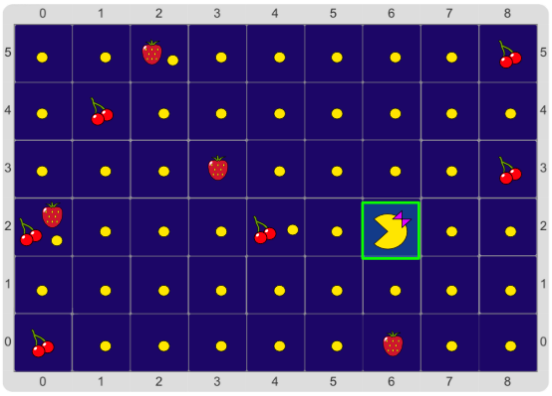
Ejercicio 10)
Para mostrar esos cuadraditos verdes con los puntos que vimos en los ejemplos anteriores se utilizó una muy útil primitiva que nos fue proporcionada:

```
procedure MostrarPuntosEnPantalla(cantidadDePuntosAMostrar)
{- PROPÓSITO:
    Muestra en la pantalla la cantidad de puntos dados en como argumento.
   PRECONDICIONES: Ninguna.
   PARÁMETROS:
    * cantidadDePuntosAMostrar: Número
        Los puntos a mostrar en la pantalla.
   OBSERVACIÓN: Los puntos se muestran como un número en un recuadro
                verde en la esquina de la celda.
-}
```

Ahora queremos asegurarnos de poder mostrar los puntos correspondientes a lo que Ms. Gobs-Man efectivamente vaya a comer en la celda actual.

Crear el procedimiento **ComerLoQueHayEnLaCeldaYMostrarPuntos** que haga que Ms. Gobs-Man coma todo lo que hay en la celda en donde está parada, y que se muestre en dicha celda los puntos que se obtienen tras comer lo que allí había.

Ejercicio 11)
Se desea saber cuántos puntos es posible obtener en un nivel determinado. Esto dependerá por supuesto de la cantidad de celdas que haya en dicho nivel, así como de que haya en cada celda (cocos, cerezas, frutillas, combinaciones de estas o nada). Se pide entonces realizar la función **cantidadDePuntosEnElNivel** que indique la cantidad total de puntos que se pueden obtener en el nivel. Por ejemplo, en el siguiente nivel se obtienen 18700 puntos (considerando que en el lugar en donde inicia Ms. Gobs-Man no hay nada). Puede asumirse que el cabezal se encuentra sobre Ms. Gobs-Man.



Atención: Para calcular los puntos no es necesario mover a Ms. Gobs-Man, sino solo el cabezal. Sin embargo, si movemos a Ms. Gobs-Man tampoco representará un problema, pues las funciones no tienen efecto, sino que describen valores.

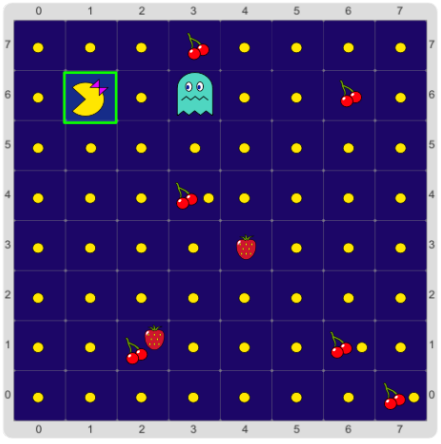
Ejercicio 12)
Es interesante poder determinar si Ms. Gobs-Man va a morir a causa de cruzarse con un fantasma o no. (Recordemos que en un nivel puede o no haber fantasmas). Se desea entonces la función **`hayAlgúnFantasmaEnElNivel`** que indica si hay un fantasma en el nivel. Por cierto, se puede asumir que el cabezal se encuentra sobre Ms. Gobs-Man.

Pista: Esta función es muy parecida a buscar un fantasma y luego morir, pero en lugar de morir debo indicar si encontré o no el fantasma. Notar que no se necesitan variables para resolver el problema.

Ejercicio 13)
Como Ms. Gobs-Man puede cruzarse con un fantasma en el camino, y en ese caso el juego termina en ese momento, los puntos totales que acumula Ms. Gobs-Man en un nivel no siempre son el total de las cosas que hay en el tablero, sino solamente aquellas que “come” hasta que encuentra el fantasma, si es que hubiera uno. En ese sentido, las direcciones hacia las cuales Ms. Gobs-Man realiza un recorrido comiendo lo que encuentra son importantes. Por ejemplo, si parte de la celda Sur-Oeste y se mueve primero al Este y luego al Norte, podría conseguir menos puntos (o más) que si parte de la celda Norte-Este y se mueve al Sur y al Oeste.

Por eso es interesante poder calcular cuantos puntos obtendrá Ms. Gobs-Man hasta toparse con un fantasma (si hubiera uno), si realiza un recorrido en dos direcciones determinadas, dadas por parámetro.
Escribir **`cantidadDePuntosEnNivelHacia_Y_`**, que dadas dos direcciones, **`direcciónPrincipal`** y **`direcciónSecundaria`**, indica cuántos puntos acumularía Ms.Gobs-Man en un recorrido en dicha dirección. Nuevamente, el cabezal arranca sobre Ms. Gobs-Man.

En el ejemplo siguiente, si el recorrido se realiza hacia el Este y el Sur (partiendo de la esquina Norte-Oeste) solo se obtendrán 2900 puntos, mientras que si se realiza hacia el Oeste y el Sur (partiendo de la esquina Norte-Este) se obtendrán 5000. Otras direcciones darán otros puntajes.



Ejercicio 14)
Se desea saber cuál de dos opciones de recorridos es más conveniente realizar. Por ejemplo, es mejor recorrer hacia el Norte y el Este, que hacia el Sur y el Este (siempre considerando mejor aquel recorrido en donde se obtienen más puntos). Para determinarlo, se pide escribir la función **`esMejorRecorridoHacia_Y_QueHacia_Y_`**, que dadas 4 direcciones, **`dirPrincipal1`**, **`dirSecundaria1`**, **`dirPrincipal2`**, **`dirSecundaria2`**, indica si un recorrido en **`dirPrincipal1`** y **`dirSecundaria1`** acumula efectivamente más puntos que un recorrido en **`dirPrincipal2`** y **`dirSecundaria2`**.

Si consideramos el ejemplo anterior, la función llamada como **`esMejorRecorridoHacia_Y_QueHacia_Y_(Este, Sur, Oeste, Sur)`** indica que es falso, pues en el recorrido Este-Sur se acumularían solamente 2900 puntos, mientras que en el Oeste-Sur serían 5000. En cambio, **`esMejorRecorridoHacia_Y_QueHacia_Y_(Oeste, Sur, Este, Sur)`** indica que es verdadero, por la misma razón.

Ejercicio 15)
Queremos también poder determinar si Ms. Gobs-Man ha logrado llegar a un punto en donde está cerca de finalizar el nivel, en particular, si completó más de la mitad del mismo.
Para esto, se pide implementar la función **`másDeLaMitadDelNivelHacia_Y_`**, que dadas 2 direcciones, **`dirPrincipal`** y **`dirSecundaria`**, indique si Ms. Gobs-Man pasó más de la mitad del nivel recorriendo hacia **`dirPrincipal`** y **`dirSecundaria`**. Notar nuevamente el cabezal está sobre Ms. Gobs-Man, y también contamos con esta útil primitiva:

```
function tamañoDelTablero()
{- PROPÓSITO: Denota el número total de celdas del tablero (nxm)
   PRECONDICIÓN: Ninguna.
-}
```

Ayuda: Tener en cuenta que Ms.Gobs-Man viene de las direcciones opuestas a aquellas hacia las cuales está recorriendo, y queremos saber cuántas celdas ya visitó.

Ejercicio 16)
El equipo de desarrollo se ha dado cuenta de que utilizar la misma representación en términos de bolitas para Ms. Gobs-Man que para Gobs-Man trae serias complicaciones. Por eso se pensó en una representación alternativa, que permita diferenciar mejor los elementos. Eso sí, algunas primitivas ahora son más complicadas y requieren operadores lógicos más complejos.

- Una bolita negra representa un coco.
- Dos bolitas negras representan una cereza.

- Tres bolitas negras en una celda indican que en la misma hay tanto un coco como una cereza.
- Una bolita roja representa una frutilla.
- Una bolita azul representa a Ms.Gobs-Man; dos, si estuviera muerta.
- Cinco bolitas azules representan un fantasma.
- Los puntos en una celda se representan con bolitas verdes (tantas como puntos).

Se pide cambiar las primitivas anteriormente realizadas en Gobs-Man para reflejar la nueva representación, así como también implementar las primitivas que son exclusivas de Ms. Gobs-Man.

ATENCIÓN: Los siguientes puntos son más avanzados, y requieren un buen manejo de las herramientas trabajadas hasta ahora. Además, requieren hacer uso de las funciones **`filaActual`** y **`columnaActual`** realizadas en la práctica. Se recomienda completar la práctica antes de continuar con los próximos ejercicios.

Ejercicio 17)
No todo es diversión al programar a Ms.Gobs-Man, porque también tenemos que programar a los malos. En este caso el cabezal se encuentra sobre un fantasma, y queremos mover al fantasma hacia donde está Ms. Gobs-Man. Para ello, debemos calcular dónde está Ms.Gobs-Man y determinar hacia donde moverse el fantasma. Para ello contamos con las siguientes primitivas:

```
procedure ParарCabezalEnMsGobsMan()
{- PROPÓSITO: Posiciona el cabezal sobre Ms. Gobs-Man.
   PRECONDICIÓN: Ms. Gobs-Man está viva en el tablero.
-}
```

```
procedure MoverFantasmaAl(dirección)
{- PROPÓSITO: Mueve al fantasma de la celda actual una celda hacia
              la dirección dada.
   PRECONDICIÓN: El cabezal se encuentra sobre un fantasma.
   PARÁMETRO:
      * dirección: Dirección - La dirección a la cual mover el fantasma.
-}
```

Realizar el procedimiento **`MoverFantasmaHaciaMsGobsMan`** que mueve el cabezal hacia Ms.Gobs-Man una celda, utilizando el siguiente criterio.

- Si Ms. Gobs-Man se encuentra en una fila y columna distinta a la de Ms.Gobs-Man, mueve el fantasma en diagonal hacia las direcciones en las que se encuentre Ms.Gobs-Man.
- Si Ms.Gobs-Man se encuentra en la misma fila que el fantasma, solo lo mueve una celda sobre la columna actual, en dirección a Ms.Gobs-Man.
- Si Ms.Gobs-Man se encuentra en la misma columna que el fantasma, solo lo mueve una celda sobre la columna actual, en dirección a Ms.Gobs-Man.

Realizar este procedimiento no es fácil, y es conveniente descomponer el problema en tareas mucho más pequeñas y simples. Es por eso que nuestro equipo de analistas ya ha planteado una serie de funciones que pueden ser útiles para solucionar el problema usando una estrategia top-down. A saber, se espera que se utilicen las siguientes funciones (y que sean implementadas también; ¿o alguien puede pensar que se programan solas?) para solucionar el procedimiento anteriormente mencionado:

- **`elFantasmaDebeMoverseDeFila`**, que indica si el fantasma no se encuentra en la misma fila que Ms. Gobs-Man.
- **`elFantasmaDebeMoverseDeColumna`**, que indica si el fantasma no se encuentra en la misma columna que Ms. Gobs-Man.

- **`direcciónEnFilaAMoverElFantasma`**, que dado que el fantasma no está en la misma fila que Ms. Gobs-Man, describe la dirección a la cual el fantasma se debería mover para quedar más cerca que Ms. Gobs-Man (Este u Oeste).
- **`direcciónEnColumnaAMoverElFantasma`**, que dado que el fantasma no está en la misma columna que Ms. Gobs-Man, describe la dirección a la cual el fantasma se debería mover para quedar más cerca que Ms. Gobs-Man (Norte o Sur).

A su vez, se recomienda realizar las siguientes funciones para solucionar las anteriores:

- **`filaDondeEstáElFantasma`**, que describe el número de fila donde se encuentra el fantasma.
- **`columnaDondeEstáElFantasma`**, que describe el número de columna donde se encuentra el fantasma.
- **`filaDondeEstáMsGobsMan`**, que describe el número de fila donde se encuentra Ms.Gobs-Man.
- **`columnaDondeEstáMsGobsMan`**, que describe el número de columna donde se encuentra Ms. Gobs-Man.

Ejercicio 18)

Se desea contar con la función **`elFantasmaSeComeráAMsGobsManAContinuación`** , que indique si, tras mover el fantasma una única vez más, este alcanzará a Ms. Gobs-Man. Puede asumirse que el cabezal está sobre el fantasma.