Práctica integradora de Recorridos

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

Gobs-Man es un clon argentino de un popular juego de arcade de principios de los 80, que está realizado en Gobstones.

En los siguientes ejercicios implementaremos diversos procedimientos y funciones que trabajan distintos aspectos del juego (no implementaremos el juego en sí, sino procedimientos y funciones que podrían servir para el mismo).

Gobs-Man es una criatura que vive en un tablero del cual desconocemos su ancho y su alto, el cual varía de un nivel a otro. Gobs-Man puede ser programado para moverse de una celda a otra del tablero, para lo cual podemos utilizar las siguientes primitivas:

```
procedure MoverGobsManAl_(dirección)
PROPÓSITO: Mueve a Gobs-Man a la celda vecina en la dirección “dirección”.
    El cabezal queda sobre Gobs-Man.
PRECONDICIÓNES:
    * Existe una celda vecina en dirección “dirección”.
    * El cabezal se encuentra sobre Gobs-Man.
PARÁMETROS:
    * dirección: Dirección - Indica hacia dónde se moverá Gobs-Man.
```

```
procedure LlevarGobsManAlBorde_(dirección)
PROPÓSITO: Mueve a Gobs-Man a la celda en el borde hacia la dirección
    “dirección”. El cabezal queda sobre Gobs-Man.
PRECONDICIÓNES:
    * El cabezal se encuentra sobre Gobs-Man.
PARÁMETROS:
    * dirección: Dirección - Indica hacia dónde se moverá Gobs-Man.
```

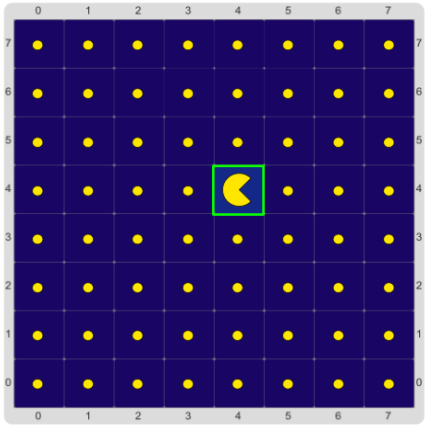
A medida que avancemos en los distintos ejercicios iremos introduciendo las reglas puntuales del juego, y nuevas primitivas que podrán utilizarse junto con las anteriormente mencionadas para solucionar el problema.

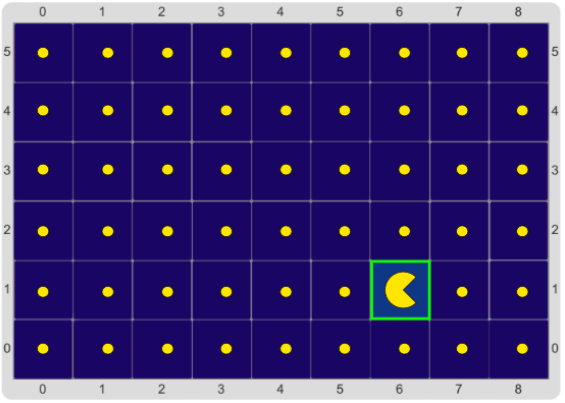
ATENCIÓN: No tiene que implementar las primitivas salvo que así lo indique el enunciado, pero si puede hacer uso de ellas para solucionar todos los problemas planteados.

Ejercicio 1)
Al comenzar un nuevo “nivel” del juego, en cada celda del tablero hay un “coco” (pequeños puntos amarillos) que son el alimento natural de los seres como Gobs-Man. El objetivo de Gobs-Man es precisamente comerse todos los cocos del nivel. Para poder hacer esto, contamos con la siguiente primitiva adicional a las anteriores:

```
procedure ComerCoco()
PROPÓSITO: Come el coco que hay en la celda actual.
PRECONDICIONES:
    * Hay un coco en la celda actual.
    * Gobs-Man está en la celda actual.
```

Se desea implementar el procedimiento **ComerTodosLosCocosDelNivel**(), que hace que Gobs-Man se coma absolutamente todos los cocos del nivel (tablero). Sabemos, *por precondición de dicho procedimiento, que hay un coco en cada celda del tablero* (incluida en la que inicia Gobs-Man). A continuación hay algunos posibles niveles de Gobs-Man (Note que son solo ejemplos, y que su solución tiene que funcionar en cualquiera de estos tableros, e incluso otros que cumplan las mismas características)



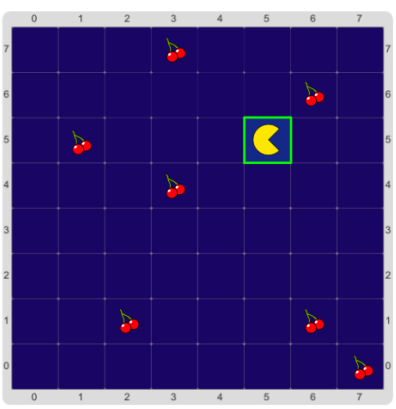


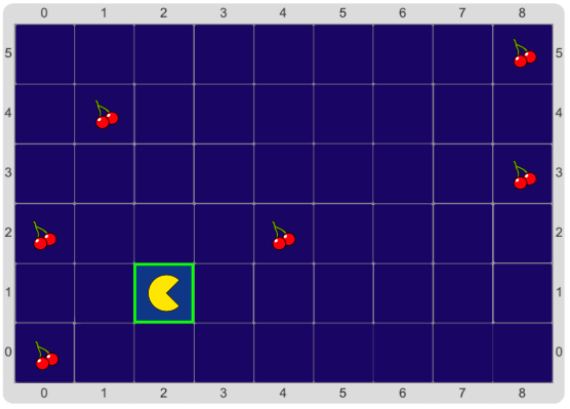
Ejercicio 2)
Gobs-Man también gusta de comer cerezas. En este caso queremos que Gobs-Man se coma absolutamente todas las cerezas del nivel. Ojo, a diferencia de los cocos, las cerezas no están en todas las celdas, sino que pueden aparecer en algunas celdas y en otras no, y nunca sabemos al arrancar un nivel en cuáles celdas estarán las cerezas. Para implementar esto necesitaremos unas nuevas primitivas:

```
procedure ComerCereza()
PROPÓSITO: Come la cereza haya en la celda actual.
PRECONDICIONES:
    * Hay una cereza en la celda actual.
    * Gobs-Man está en la celda actual.
```

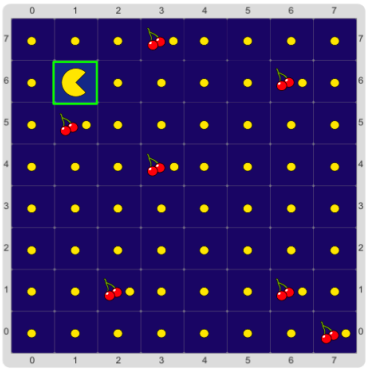
```
function hayCereza()
PROPÓSITO: Indica cuando hay una cereza en la celda actual.
PRECONDICIÓN: Ninguna.
```

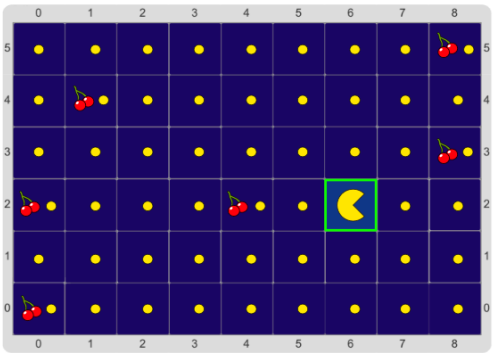
Ahora si, implementemos el procedimiento **ComerTodasLasCerezasDelNivel**(), que hace que Gobs-Man se coma todas las cerezas del tablero. Note que los tableros iniciales posibles de este ejercicio no tienen cocos, sino que en cada celda puede haber una cereza, o no haber nada, como muestran los ejemplos de tableros iniciales a continuación:





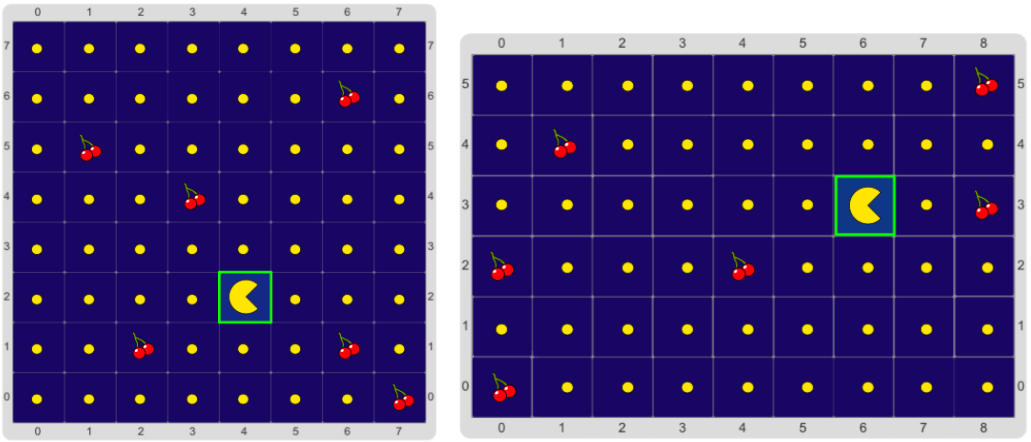
Ejercicio 3)
Para este ejercicio queremos trabajar sobre niveles que tienen tanto cerezas como cocos. En nuestros tableros iniciales habrá cocos en todas las celdas, y en algunas habrá adicionalmente una cereza. Gobs-Man quiere comerse absolutamente todo lo que encuentre en el nivel, y nosotros debemos determinar cómo se realiza entonces el código de **`ComerTodoLoQueSeEncuentreEnElNivel`**().





Pista: Si su solución requiere más de dos líneas de código, plantee otra estrategia (piense en qué cosas ya realizó anteriormente)

Ejercicio 4)
El programador del juego ha decidido hacer unos pequeños retoques a cómo inician los niveles. Ahora, en todas las celdas hay cocos, menos en aquellas donde hay una cereza. Es decir, en cada celda puede, o bien haber una cereza, o bien haber un coco (solo estará vacía la celda cuando Gobs-Man se haya comido todo lo de la celda, nunca al inicio del nivel). Debemos entonces volver a realizar **`ComerTodoLoQueSeEncuentreEnElNivel`**(), y en este caso, la solución no es tan sencilla como en el ejercicio anterior. A continuación, algunos posibles tableros iniciales de este caso:



ATENCIÓN: Note que no dispone de una primitiva "hayCoco" para solucionar el problema. Si su estrategia está realizada utilizando dicha primitiva, entonces es incorrecta.

ATENCIÓN: Si a esta altura sus soluciones le están demandando el planteo de más de un recorrido, probablemente esté planteando recorridos sobre filas o columnas. Le proponemos plantee la solución en términos de un recorrido único sobre todas las celdas del tablero, o los siguientes ejercicios se volverán sumamente complicados.

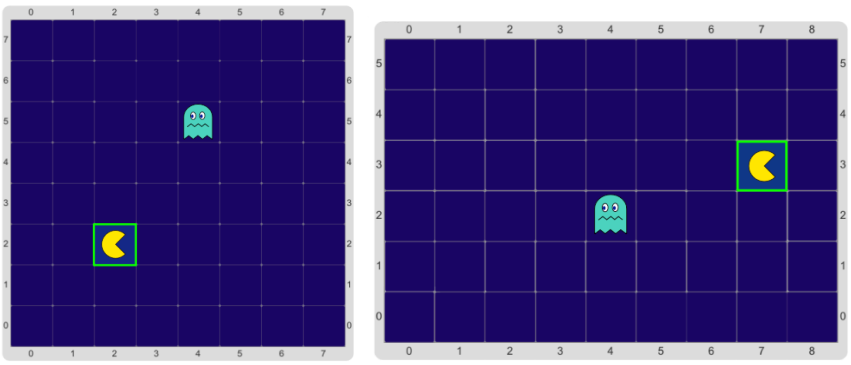
Ejercicio 5)
Gobs-Man puede toparse en algún momento con un fantasma. Si lo hace, Gobs-Man sufre un paro cardíaco que la hace morir en la celda en donde vió el espectro. Sabiendo que existen ahora las siguientes primitivas:

```
procedure MorirGobsMan()
PROPÓSITO: Hace que Gobs-Man muera, dejando su cuerpo en la celda actual.
PRECONDICIONES:
    * El cabezal se encuentra sobre Gobs-Man
```

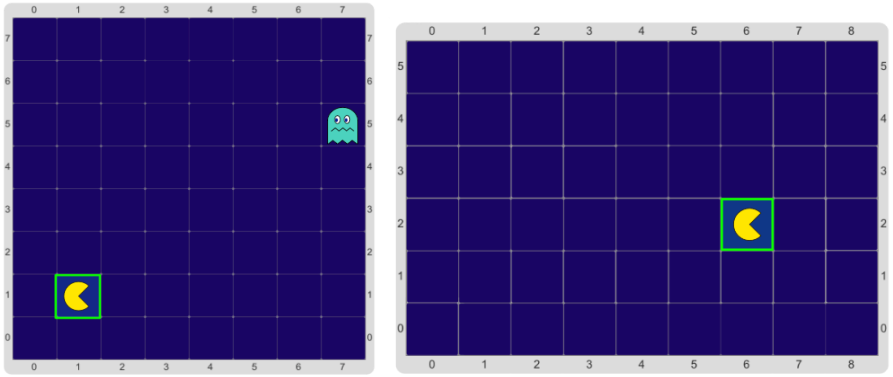
```
function hayFantasma()
PROPÓSITO: Indica cuando hay un fantasma en la celda actual.
PRECONDICIÓN: Ninguna.
```

ATENCIÓN: Note que una vez muerto Gobs-Man no puede moverse. Es decir, los procedimientos que mueven a Gobs-Man tienen ahora una nueva precondición: Gobs-Man está vivo.

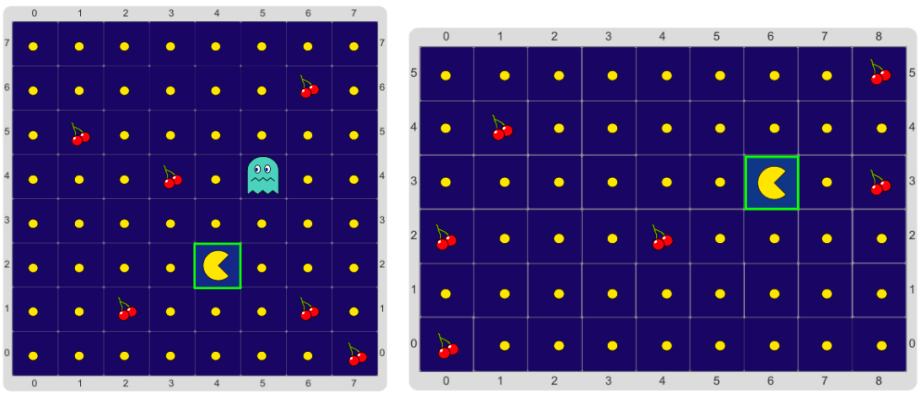
Se pide entonces hagamos una prueba sobre un nivel vacío (Es decir, en las celdas no hay cocos ni cerezas) donde Gobs-Man deberá moverse desde la celda más al Oeste y al Sur, hacia la celda más al Norte y al Este. Se garantiza que en algún lado del tablero habrá un fantasma, y Gob-Man debe morir en la celda en donde encuentre el mismo. Realice entonces el procedimiento **`RecorrerNivelMuriendoEnElFantasma()`.** Como es costumbre, dejamos algunos tableros iniciales:



Ejercicio 6)
Si bien hemos logrado que Gobs-Man muera en el lugar correcto, también se desea contemplar los niveles en donde tal vez no haya un fantasma. Es decir, ahora queremos volver a recorrer el nivel, pero esta vez, no tenemos la certeza de que hay un fantasma en el nivel. Si hay uno, Gobs-Man deberá morir allí, sino, Gobs-Man deberá quedar vivo en la última celda del recorrido. Realice entonces el procedimiento **`RecorrerNivelMuriendoSiHayFantasma()`** que solucione dicho problema. Los tableros iniciales son idénticos a los anteriores, pero, el fantasma podría no estar, como muestra el segundo tablero de ejemplo:



Ejercicio 7)
En este ejercicio queremos integrar todas nuestras soluciones al momento. Aunque probablemente no podamos reutilizar el código, si podremos reutilizar las ideas de lo que venimos trabajando.
En este caso, el nivel comienza con un coco en cada celda, menos en las que hay cerezas, y tal vez, algún fantasma en alguna celda del tablero. Gobs-Man debe comer todos los cocos y cerezas que pueda, partiendo esta vez de la esquina Norte y Oeste, y yendo hacia el Sur y el Este. Al finalizar el nivel, Gobs-Man debe quedar en dicha esquina, si es que no se cruzó con ningún fantasma. Si por el contrario el nivel tiene un fantasma, Gobs-Man deberá comer todo lo que tenga en el camino, hasta que se tope con el espectro, donde morirá y terminará el juego. Implemente entonces **`JugarNivel`**() que realice lo mencionado.



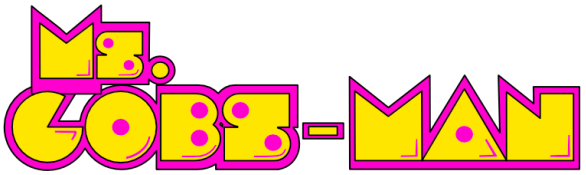
Ejercicio 8)
Que pasa si quiero que Gobs-Man ahora recorra en sentido inverso, es decir, partiendo en la esquina Sur y

Este, y yendo hacia el Norte y el Oeste. ¿Su solución permite cambiar fácilmente ese detalle? Si la respuesta es negativa, piense alguna forma de lograrlo. Pista, el truco está en pasar información a los procedimientos que realizan el recorrido celda a celda.

Ejercicio 9)
Queremos realizar el juego, y probar que funcionen nuestras soluciones, pero el diseñador gráfico ha renunciado y no tenemos vestimentas ni primitivas que nos abstraiga de la representación, debemos contentarnos con ver bolitas. Por suerte, todas nuestras soluciones anteriores asumen la existencia de ciertos procedimientos y funciones primitivas, por lo que bastará implementar las mismas para tener andando nuestro trabajo previo. Asumiremos la siguiente representación:

- Gobs-Man estará representado por una bolita de color Azul si está vivo, y dos, si está muerto.
- Un coco estará representado por una bolita de color Negro.
- Una cereza estará representada por dos bolitas de color Rojo.
- Un fantasma estará representado por cinco bolitas de color Verde.

Se pide entonces implemente cada uno de los procedimientos y funciones primitivas mencionados en esta guía utilizando esta representación, y luego pruebe las soluciones que hicimos en papel, en la máquina.



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.

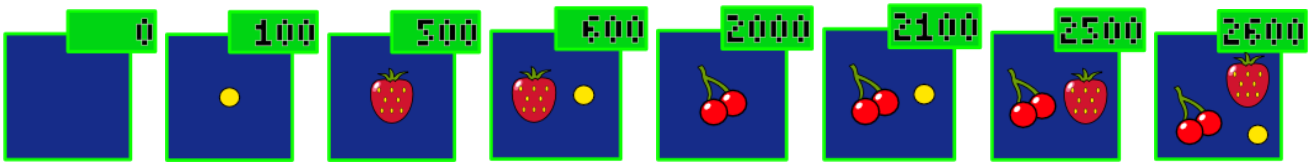
Esta vez el jugador se pondrá en la piel de Ms.Gobs-Man, la versión femenina de nuestro aclamado heroe comecocos. Como en la primera versión del juego, el objetivo será lograr que Ms. Gobs-Man se coma todos los cocos y las cerezas. Para ello contaremos con todas las primitivas que se presentaron en el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

```
function hayCoco()
PROPÓSITO: Indica cuando hay un coco en la celda actual.
PRECONDICIÓNES: Ninguna.
```

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

Ejercicio 10)
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. Note 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. Escriba 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 11)
Se nos plantea ahora que además de cerezas y cocos, las celdas pueden contener frutillas (O fresas, si prefiere). Las frutillas otorgan 500 puntos solamente, pero pueden encontrarse en cualquier celda, por lo que ahora tenemos las siguientes posibilidades:



Debe replantear la función **`puntajeAObtenerEnCeldaActual()`** para tener en cuenta dicha situación. Si su

estrategia anterior fue buena, entonces este cambio no debería redundar en demasiado trabajo. Si por el contrario la solución no fue buena, le llevará más esfuerzo (Si fuera este último caso, lo invitamos a repensar si está separando el problema en las subtareas correctas, o ver si puede realizar más). Ah, por cierto, casi se nos olvida, también cuenta con la primitiva siguiente:

```
function hayFrutilla()
PROPÓSITO: Indica cuando hay una frutilla en la celda actual.
PRECONDICIÓNES: Ninguna.
```

Atención: Piense si su solución escala correctamente si hubieran otras tres posibles frutas (ej. naranjas, bananas y manzanas)

Ejercicio 12)
Por cierto, para mostrar esos cuadraditos verdes con los puntos que vimos en los ejemplos anteriores se utilizó una muy útil primitiva que nos proporcionaron:

```
procedure Mostrar_PuntosEnPantalla(cantidadDePuntosAMostrar)
PROPÓSITO: Muestra en la pantalla la cantidad de puntos dados en como
    argumento.
OBSERVACIÓN: Los puntos se muestran como un número en un recuadro verde
    en la esquina de la celda.
PRECONDICIÓNES:
    * No debe haber elementos para comer en la celda actual.
    * El cabezal se encuentra sobre Ms. Gobs-Man.
PARÁMETROS:
    * cantidadDePuntosAMostrar: Número - Los puntos a mostrar en la pantalla.
```

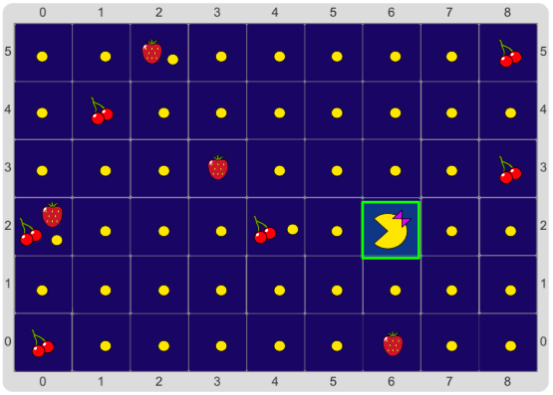
Ahora queremos asegurarnos de poder mostrar los puntos correspondientes a lo que Ms. Gobs-Man efectivamente vaya a comer en la celda actual.

Se espera que usted cree 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.

Probablemente necesite también del procedimiento siguiente como primitiva:

```
procedure ComerFrutilla()
PROPÓSITO: Come la frutilla de la celda actual.
PRECONDICIÓNES:
   * Hay una frutilla en la celda actual.
   * El cabezal se encuentra sobre Ms. Gobs-Man
```

Ejercicio 13)
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 realice 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 asumir que el cabezal se encuentra sobre Ms. Gobs-Man.



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

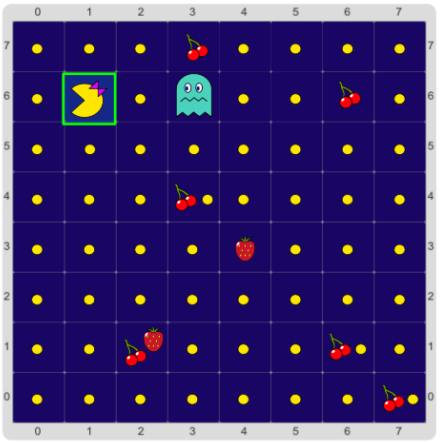
Ejercicio 14)
Es interesante poder determinar si Ms. Gobs-Man va a morir a causa de cruzarse con un fantasma o no (Recordemos que en un nivel puede o no haber fantasmas). Se desea entonces la función **`hayAlgúnFantasmaEnElNivel`**( ) que indica si hay un fantasma en el nivel. Por cierto, puede asumir 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. Note que no necesita variables para resolver el problema.

Ejercicio 15)
Cómo Ms. Gobs-Man puede cruzarse con un fantasma en el camino, y en ese caso, el juego termina en ese momento. Es decir, 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 es importante. 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 poner un ejemplo.

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.
Se pide escriba **`cantidadDePuntosEnNivelHacia_Y_(direcciónPrincipal, direcciónSecundaria)`**, una función que dadas dos direcciones 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 16)
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 que escriba la función **`esMejorRecorridoHacia_Y_QueHacia_Y_(dirPrincipal1, dirSecundaria1, dirPrincipal2, dirSecundaria2)`** que 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)`** describe **`Falso`**, pues en el recorrido Este Sur se acumulaban solo 2900 puntos, mientras que en el Oeste-Sur eran 5000. **`esMejorRecorridoHacia_Y_QueHacia_Y_(Oeste, Sur, Este, Sur)`** describe **`Verdadero`** por la misma razón.

Ejercicio 17)
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 le pide implementar la función **`masDeLaMitadDelNivelSiVaHacia_Y_(dirPrincipal, dirSecundaria)`** que indica si Ms. Gobs-Man pasó más de la mitad del nivel recorriendo hacia dirPrincipal y dirSecundaria. Note que sabemos nuevamente que 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: Tené 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 18)
El equipo de desarrollo se ha dado cuenta de que al 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 cambie cree 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 haga uso de las funciones **`filaActual`**() y **`columnaActual`**() realizadas en la práctica. Si aún no finalizó la práctica, le recomendamos la termine previo a continuar.

Ejercicio 19)
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.
```

Se pide entonces que realice el procedimiento **`MoverFantasmaHaciaMsGobsMan`**() que mueve el fantasma 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 ese 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 serle útiles para solucionar el problema usando una estrategia top-down. A saber, se espera utilice estas funciones (y las implemente) para solucionar el procedimiento anteriormente mencionado:

- **`elFantasmaDebeMoverseHorizontalmente`**() que indica si el fantasma no se encuentra en la misma columna que Ms. Gobs-Man.
- **`elFantasmaDebeMoverseVerticalmente`**() que indica si el fantasma no se encuentra en la misma fila que Ms. Gobs-Man.
- **`direcciónHorizontalAMoverElFantasma`**() 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 en términos de columnas (Este u Oeste).
- **`direcciónVerticalAMoverElFantasma`**() 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 en términos de filas (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 20)
Se desea saber con la función **`elFantasmaSeComeráAMsGobsManAContinuación`**() que indica si, tras mover el fantasma una única vez más, este alcanzará a Ms. Gobs-Man. Puede asumir que el cabezal está sobre el fantasma.