Gobstoban

A partir de los últimos conocimientos aprendidos en el equipo de desarrollo, se decidió plantear de cero el desarrollo de Gobstoban, descartando todo lo previamente realizado (por lo que las primitivas, funciones y procedimientos anteriores ya no pueden usarse). Repasemos el enunciado y veamos cómo ha cambiado.

El Sokoban (encargado de depósito, en japonés) es un videojuego de tipo puzzle desarrollado originalmente por Hiroyuki Imabayashi en 1981. El juego consiste en manipular a un personaje (el encargado) que se mueve a través de un depósito, empujando cajas que deben ser colocadas en lugares específicos del depósito.

En este examen vamos a trabajar con una adaptación propia del juego, implementada en Gobstones, a la que llamaremos Gobstoban. Primero explicaremos en detalle las reglas del juego, y luego su representación en Gobstones.

El juego se desarrolla en una grilla rectangular de tamaño variable, que representa la totalidad del depósito. En alguna ubicación del depósito se encuentran los encargados que son los personajes que un jugador manipula. Pueden haber más de uno en el tablero, para soportar modo multijugador, e incluso no haber ninguno, para soportar modo de edición del tablero.

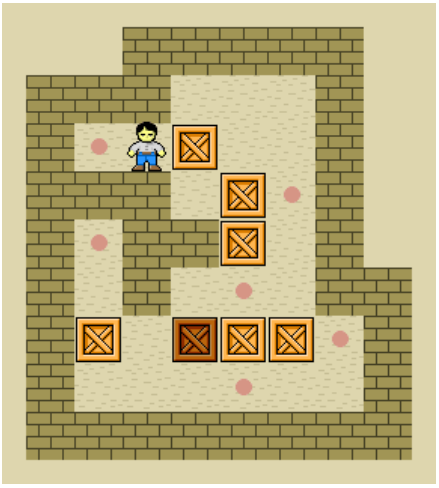
También pueden haber cajas dentro del depósito, las cuales pueden ser movidas de una ubicación a otra por el encargado, empujandolas. Además, en ciertas ubicaciones pueden haber paredes, que no pueden ser atravesadas ni por el encargado ni por las cajas. Las ubicaciones de esos elementos son excluyentes entre sí, es decir, no puede haber una caja dónde está el encargado, ni puede haber cajas sobre paredes, ni el empleado puede estar sobre una pared.

Adicionalmente, en otras ubicaciones hay marcadores de meta. Un marcador de meta es una ubicación en el depósito (que no puede coincidir con una pared) y a donde debe ubicarse una caja para lograr superar el juego (o el nivel mejor dicho). Sobre una meta sí puede colocarse una caja, o pararse el encargado.

El objetivo del juego es entonces mover al encargado por el depósito, para que empuje las cajas, llevando una sobre cada marcador de meta en el depósito.

Una caja puede ser empujada cuando el encargado se para a su lado, y solo se empuja en dirección contraria a la del encargado. Más aún, para poder empujar la caja en dicha dirección, debe haber una ubicación del depósito que no contenga ni pared, ni otra caja.

A continuación hay una muestra con vestimentas del juego, en donde se resuelve un nivel en particular.



Para programar en Gobstones este juego, se han desarrollado una serie de tipos, funciones y procedimientos que pueden usarse como primitivas.

```
type ObjetoDelJuego is variant {
    // PROPÓSITO: Modela el tipo de un objeto posible en una ubicación.
    case Encargado {}
    case Caja      {}
    case Meta      {}
    case Pared     {}
    case Vacío     {}
}
```

```
objetoAcá
PROPÓSITO: Describe el objeto en la ubicación actual.
TIPO: ObjetoDelJuego
PRECONDICIONES: Ninguna
```

```
SacarObjeto_
PROPÓSITO: Saca el objeto dado de la ubicación actual.
PARÁMETROS:
    * objetoASacar: ObjetoDelJuego - El objeto a sacar
PRECONDICIONES:
    * El objeto en la celda actual no puede ser Vacío.
    * En la celda actual está el objeto dado.
```

```
PonerObjeto_
PROPÓSITO: Pone el objeto dado en la ubicación actual.
PARÁMETROS:
    * objetoAPoner: ObjetoDelJuego - El objeto a poner
PRECONDICIONES:
    * El objeto dado no puede ser Vacío.
    * Si es una pared, la ubicación actual está vacía.
    * Sí es una caja, la ubicación actual está vacía o tiene una marca de meta.
    * Sí es un encargado, la ubicación actual está vacía o tiene una marca de
meta.
    * Sí es una meta, la ubicación actual está vacía.
```

```
objetoDebajoDe_Acá
PROPÓSITO: Describe el objeto en la ubicación actual que está debajo del objeto
dado.
TIPO: ObjetoDelJuego
PRECONDICIONES:
    * El objeto dado puede ser solo Caja o Encargado.
```

En una misma ubicación entonces puede haber solo un objeto, o pueden haber dos en los siguientes casos:

- Sí hay una caja, debajo puede estar vacío o puede haber una meta.
- Sí hay un encargado, debajo puede estar vacío o puede haber una meta.
- No puede haber dos elementos en ningún otro caso.

Notar que una ubicación con elemento vacío no necesariamente se corresponde con una celda vacía.

Actividades:

Ejercicio 1)
Realice la función **cantidadDeCajasRestantesAPonerSobreMeta** que describe la cantidad de cajas en el nivel que aún no están sobre una meta.

Ejercicio 2)
Realice la función **hayEncargadoSobreMetaEnNivel** que indica si en el nivel hay un encargado y el mismo está sobre alguna una meta.

Ejercicio 3)
Realice la función **hayCajaSobreMetaAlrededor** que indica si en alguna ubicación lindante hay una caja que esté sobre una meta. Debe resolver este problema sin uso de variables.

Ejercicio 4)
Se desea poder saber en todo momento el estado del juego general, para lo que se elaboró el siguiente tipo:

```
type StatusNivel is record {
    // PROPÓSITO: Modela el estado del nivel de Gobstoban.
    // INVARIANTE DE REPRESENTACIÓN: dificultad y cajasPendientes
                                     son >= 0.
    field dificultad                  // Número
    field esConsistente               // Booleano
    field cajasPendientes             // Número
}
```

Se pide que realice la función **statusDelJuego** que describe el estado del nivel, y donde:

- La dificultad está dada por la cantidad de metas por cada encargado en el nivel. Sí no hay encargados, la dificultad es cero.
- El nivel es consistente si hay al menos tantas marcas de meta como cajas.
- Las cajas pendientes son aquellas que no están aún sobre una meta.

Tenga en consideración que pueden haber elementos debajo de otros.