Trabajo Práctico Nº 2. Programación Lógica.

Puntuación

Puntaje Total: 100 puntos.
Aprobación: 60 puntos.
Plazo de entrega: 01/11/2022.

Condiciones de entrega

1. El presente trabajo práctico deberá resolverse en grupo de hasta 3 (tres) personas.
2. Entrega: Se realizará por medio del campus virtual de la UTN, en la tarea correspondiente al TP Nº 2. La extensión del archivo será .zip o .rar, de acuerdo al programa de compresión usado. El nombre del archivo se consigue concatenando un prefijo del número del TP con los apellidos de los integrantes separados por guiones (Ej: Pérez y Abdala, el nombre será `tp2-abdala-perez.zip`). Note que no hay espacios en blanco ni acentos en el nombre de archivo. Dentro del archivo de entrega, deben constar los siguientes:
   - Fuentes SWI-Prolog: Se debe entregar un archivo denominado `tp2.pl`.
   - Los casos de prueba se entregarán en un archivo de texto, no deben ser capturas de pantalla. Deberán cubrir diferentes resultados que puedan obtenerse de la evaluación de los predicados solicitados. Se enfatiza que se adjunten casos de prueba que sean claros, válidos y suficientes para poder probar el trabajo. Entregar este archivo con el nombre `casos-de-prueba.txt`.
   - Archivo de texto (`integrantes.txt`) con una línea para cada integrante en la cual figure el nombre del alumno/a y su dirección de email.
4. Penalización por entrega fuera de término: Si el trabajo práctico se entrega después de la fecha indicada, y hasta una semana tarde, tendrá una quita de 15 puntos. No serán recibidos trabajos luego de una semana de la fecha de entrega. Los trabajos que se deban rehacer/corregir tendrán una quita de 30 puntos.

1. Descripción del Problema

Durante la década de 1970 se introdujo un nuevo género de video juego denominado Snake, y en 1978 la primera versión para PC con el nombre de Worm. En los últimos años, a partir de mejoras gráficas, fue desarrollado en las nuevas consolas de juegos y dispositivos móviles.
El juego consiste en un worm (gusano) que se desplaza en un plano realizando movimientos individuales a una posición adyacente. El objetivo del juego es que el worm "coma" los alimentos que aparecen en el plano. Es suficiente con que la cabeza alcance la posición del alimento, sin necesidad de realizar ninguna otra acción. Con cada comida el jugador suma puntos y continúa con la siguiente.
El juego se da por perdido (hay una única vida) en algunas de estas situaciones:

i. El worm “choca” con alguna de las paredes, es decir, excede los límites del plano.
ii. El worm “choca” contra su cola, es decir, la cabeza avanza a una posición que actualmente está ocupada por él mismo.
iii. El worm “choca” contra un obstáculo. Los obstáculos son fijos desde el comienzo del juego.

Adicionalmente se pueden incorporar puertas o pasajes que permiten al worm “salir” del tablero e “ingresar” por el otro extremo (aquí también debe pasar el cuerpo completo).
A continuación se presentan movimientos del worm en un plano de 11x11. Las numeraciones no forman parte del juego, son únicamente ilustrativas. La cabeza del worm está indicada con **H**, los obstáculos con # y las puertas con **<**.

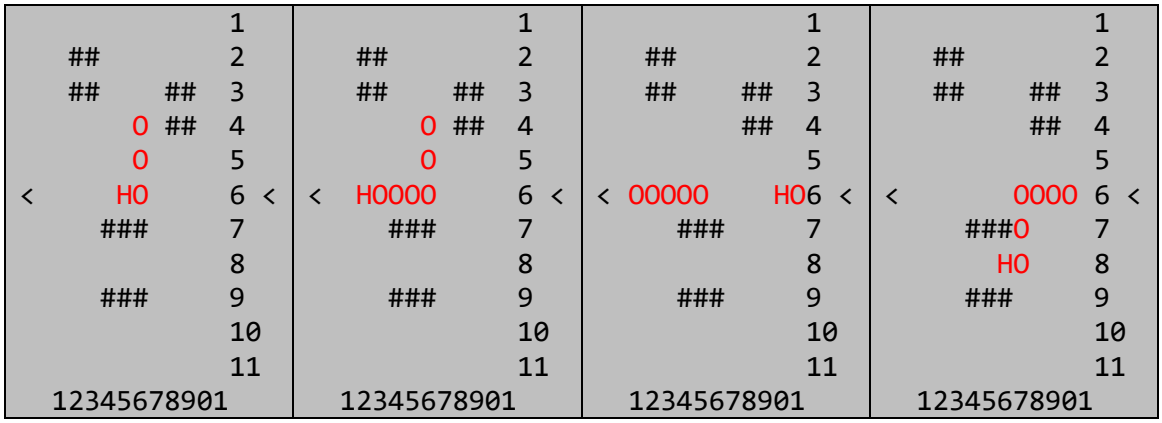

Figura 1. Movimientos del worm.

En la Figura 1 (a) el worm alcanza una comida y luego de varios movimientos se observa que creció y se dirige a la puerta en la fila 6 (b). En (c) el worm está cruzando el tablero. Finalmente, terminó de cruzar y atraviesa el espacio entre los obstáculos (d).

Se solicita escribir un programa lógico en Prolog para implementar el juego Worm de acuerdo a las reglas planteadas y las funcionalidades indicadas a continuación.

2. Funciones

2.a. worm/1

Representa un worm y las posiciones que ocupa en el plano.

`worm(L)`: L unifica con una lista, donde cada elemento es una lista de dos términos: fila y columna. El primer elemento de L siempre es la cabeza del worm. Ejemplos:

`2.a.1) worm([[2,3],[2,4],[2,5]])`

Corresponde a un worm de longitud tres con cabeza en la fila 2 y columna 3.

`2.a.2) worm([[2,5],[2,4],[2,3]])`

`2.a.3) worm([[1,5],[2,5],[3,5]])`

2.b. **comida/1**

Representa las comidas que están en el plano y aún no fueron ingeridas.

`comida(L)`: L unifica con una lista de comidas. Cada elemento de L es una lista de dos términos: fila y columna.

2.c. **juego/3**

Esta función representa un juego. Cada juego incluye un worm y una lista de comidas aún no ingeridas.

`juego(W,L)`: W unifica con una función worm y L con una función comida, de acuerdo a las definiciones previas. Ejemplos:

`2.c.1) juego(worm([[2,3],[2,4],[2,5]]),comida([[10,10],[4,2]])`

Corresponde a un juego en el que hay dos comidas pendientes.

`2.c.2) juego(worm([[5,1],[6,1],[7,1],[8,1],[9,1]]),comida([])`

Corresponde a un juego en el que no hay comidas pendientes.

3. Predicados Disponibles

En el Anexo se encuentran las definiciones para los siguientes predicados.

3.a. **plano/2**

Se empleará para representar las dimensiones del plano de juego.

`plano(N,M)`: N unifica con la cantidad de filas y M con la cantidad de columnas del plano.

3.b. **obstaculos/1**

Se empleará para representar las posiciones en donde se encuentran los obstáculos del juego.

`obstaculos(L)`: L unifica con una lista donde cada elemento de la misma es una lista con la fila y columna del obstáculo.

3.c. **puerta/3**

Se empleará para representar las puertas del juego.

`puerta(E,S,D)`: E unifica con la posición de entrada a la puerta, S con la nueva posición de salida y D con la dirección de un avance.

3.d. **wormCrecimiento/1**

Se empleará para indicar el crecimiento del worm luego de comer.

`wormCrecimiento(N)`: N unifica con un número que indica la longitud de crecimiento del worm.

3.e. **puntaje/1**

Se empleará para indicar el puntaje obtenido luego de ingerir una comida.

`puntaje(P)`: P unifica con un número que indica el puntaje obtenido.

4. Predicados a Implementar

Se solicita implementar los siguientes predicados.

4.a. **avanzar/3**

Se empleará para el avance a una única posición adyacente del worm. El avance implica agregar la nueva posición al comienzo y eliminar la última.

`avanzar(W1,W2,D)`: W1 y W2 unifican con funciones worm y D con la dirección de avance del worm: a, b, d, i. Siendo, a = arriba, b = abajo, i = izquierda y d = derecha. W2 debe ser el worm resultante luego del avance de W1 con dirección D. Ejemplos:

```
4.a.1) ?- avanzar(worm([[1,2],[1,3],[1,4]]),W2,D).
W2 = worm([[1, 1], [1, 2], [1, 3]]), D = i ;
W2 = worm([[2, 2], [1, 2], [1, 3]]), D = b ;
W2 = worm([[1, 3], [1, 2], [1, 3]]), D = d ;
W2 = worm([[0, 2], [1, 2], [1, 3]]), D = a ;
false.
4.a.2) ?- avanzar(worm([[6,1],[6,2],[6,3]]),W2,i).
W2 = worm([[6, 0], [6, 1], [6, 2]]) ;
W2 = worm([[6, 11], [6, 1], [6, 2]]) ;
false.
```

En la segunda solución el worm avanza hacia la izquierda y comienza a atravesar una puerta con entrada en [6, 1] y salida en [6, 11].

Nota: No todas las soluciones corresponden a posiciones válidas. Los avances con dirección derecha y arriba en ejemplo 4.a.1 y una solución del ejemplo 4.a.2 muestran posiciones no permitidas. No realizar controles en este predicado.

4.b. **seguro/1**

Se empleará para indicar si la nueva posición de un worm es segura.

`seguro(J)`: J unifica con la función juego en el que el worm cumple las siguientes condiciones:

I. Permanece dentro del plano.
II. No ocupa la posición de un obstáculo.
III. No vuelve sobre sí mismo.

Las unificaciones no válidas de los ejemplos de 4.a.1 y 4.a.2 no cumplen con alguna/s de estas condiciones.

4.c. movimiento/4

Se empleará para el movimiento del worm a una nueva posición (no necesariamente adyacente). Usar avance/3 y seguro/1.

`movimiento(J,J2,N,LD)`: J y J2 unifican con funciones juego, N con la cantidad de avances realizados y LD con una lista de constantes con las direcciones de los avances. J2 es el juego resultante de mover el worm de J con las direcciones de LD.

Importante: Un movimiento incluye N avances (N > 0). Cada avance debe ser seguro antes de realizar el siguiente. Ejemplos:

4.c.1)

```
?- movimiento(juego(worm([[1,1],[1,2],[1,3]]),comida([11,11])),
J,1,L).
J = juego(worm([[2, 1], [1, 1], [1, 2]]), comida([11, 11])),
L = [b] ;
false.
```

Dado que el worm está en el extremo superior derecho con la cabeza en la posición [1,1], el único movimiento seguro es con dirección hacia abajo.

4.c.2)

```
?- movimiento(juego(worm([[1,1],[1,2],[1,3]]),comida([11,11])),J,2,L).
J = juego(worm([[3, 1], [2, 1], [1, 1]]), comida([11, 11])),
L = [b, b] ;
false.
```

Luego del primer movimiento hacia abajo, el único movimiento seguro es también con dirección hacia abajo (a su izquierda está el límite del plano y a derecha hay un obstáculo).

4.c.3)

```
?- movimiento(juego(worm([[5,5],[5,6],[5,7]]),comida([11,11])),
J,2,L).
```

```
J = juego(worm([[5, 3], [5, 4], [5, 5]]), comida([11, 11])),
L = [i, i] ;
J = juego(worm([[6, 4], [5, 4], [5, 5]]), comida([11, 11])),
L = [i, b] ;
J = juego(worm([[4, 4], [5, 4], [5, 5]]), comida([11, 11])),
L = [i, a] ;
j = juego(worm([[6, 4], [6, 5], [5, 5]]), comida([11, 11])),
L = [b, i] ;
J = juego(worm([[6, 6], [6, 5], [5, 5]]), comida([11, 11])),
L = [b, d] ;
J = juego(worm([[4, 4], [4, 5], [5, 5]]), comida([11, 11])),
L = [a, i] ;
J = juego(worm([[4, 6], [4, 5], [5, 5]]), comida([11, 11])),
L = [a, d] ;
J = juego(worm([[3, 5], [4, 5], [5, 5]]), comida([11, 11])),
L = [a, a] ;
false.
```

El worm tiene varias posibilidades de movimientos. El primer avance no incluye ninguna dirección hacia la derecha porque estaría volviendo sobre sí mismo (un análisis similar puede hacerse con el segundo avance).

4.d. intento/4

Se empleará para intentar movimientos del worm a una nueva posición controlando la cantidad de avances realizados. El objetivo es realizar un movimiento a una nueva posición con la menor cantidad de avances posibles. Usar movimiento/4.

`intento(J,J2,N,LD)`: J y J2 unifican con funciones juego, N con la cantidad de movimientos a partir de la cual se comenzará a intentar el movimiento y LD unifica con la lista de constantes con las direcciones. J2 es el juego resultante de mover el worm de J con las direcciones de LD. Ejemplos:

4.d.1)

```
?- intento(juego(worm([[5,5],[5,6],[5,7]]),comida([11,11])),
J,1,L).
J = juego(worm([[5, 4], [5, 5], [5, 6]]), comida([11, 11])),
L = [i] ;
J = juego(worm([[6, 5], [5, 5], [5, 6]]), comida([11, 11])),
```

```
L = [b] ;
J = juego(worm([[4, 5], [5, 5], [5, 6]]), comida([11, 11])),
L = [a] ;
J = juego(worm([[5, 3], [5, 4], [5, 5]]), comida([11, 11])),
L = [i, i] ;
J = juego(worm([[6, 4], [5, 4], [5, 5]]), comida([11, 11])),
L = [i, b] .....
```

Intenta movimientos comenzando con un avance, luego movimientos de dos avances y así sucesivamente.

4.d.2)

```
?- intento(juego(worm([[5,5],[5,6],[5,7]]),comida([[11,11]])),
 juego(worm([[3,5]|_]),_),1,L).
L = [a, a] ;
L = [i, a, d, a] ;
L = [i, a, a, d] ;
L = [a, i, a, d] ;
L = [a, d, a, i] ;
L = [i, i, a, d, d, a] ;
L = [i, i, a, d, a, d] .....
```

En este ejemplo se busca que la cabeza del worm luego del intento esté en la posición [3,5], comenzando con movimientos de 1 avance, luego 2 avances, etc. Se observa que con 1, 3 y 5 avances no es posible alcanzar la nueva posición del worm.

4.d.3)

```
?- intento(juego(worm([[5,5],[5,6],[5,7]]),comida([[11,11]])),
J,1,[a,a,a,a]).
J = juego(worm([[1, 5], [2, 5], [3, 5]]), comida([[11, 11]])) ;
false.
```

En este ejemplo se buscan 4 avances, todos con dirección arriba. Observe que se comienza con intentos de 1 movimiento y continúa hasta hallar la solución.

4.d.4)

```
?- intento(juego(worm([[5,5],[5,6],[5,7]]),
comida([[11,11]])),J,5,[a,a,a,a]).
false.
```

No hay unificación porque no es posible realizar 4 avances partiendo de 5 movimientos.

4.e. creceWorm/2

Se empleará para el crecimiento del worm.

`creceWorm(W1,W2)`: W1 y W2 unifican con funciones worm. W2 es la resultante de W1 luego del crecimiento.

Importante: Para representar el crecimiento debe repetirse N veces la última posición (usar predicado 3.d para obtener N). De esta forma la lista de posiciones tendrá N elementos adicionales y en los próximos avances los repetidos se eliminarán. Ejemplo:

```
4.e.1) ?- creceWorm(worm([[5,5],[5,6],[5,7]]),W).
W = worm([[5, 5], [5, 6], [5, 7], [5, 7], [5, 7], [5, 7]])
```

4.f. jugar/4

Se empleará para jugar de forma autónoma. El juego finaliza cuando el worm ingirió todas las comidas de la lista.

`jugar(J,JF,L,P)`: J y JF unifican con funciones juego, P con el puntaje obtenido y L con una lista, donde cada elemento es una lista de direcciones para alcanzar una comida (si el juego inicial posee n comidas, L deberá tener n listas de direcciones). JF es el juego resultante de ingerir todas las comidas en J.

Ejemplos:

```
?- W = worm([[6,2],[6,3],[6,4]]),
jugar(juego(W,comida([[5,5],[11,6]])),J,L,P).
L = [[a, d, d, d], [d, a, a, a, a, a]],
J = juego(worm([[11,6],[1,6],[2,6],[3,6],[4,6],[5,6],[5,6],[5,6],
[5,6]]),comida([]))
P = 20.
?- W = worm([[6,2],[6,3],[6,4]]),
jugar(juego(W,comida([[5,5],[11,6],[1,11],[1,9],[1,5]])),J,L,P).
L = [[a, d, d, d], [d, a, a, a, a, a], [d, d, d, d, d, a, a, a, a, a,
a, a, a, a, a], [i, i], [i, i, i, i]],
J = juego(worm([[1,5],[1,6],[1,7],[1,8],[1,9],[1,10],[1,11],[2,11],
[3,11],[4,11],[5,11],[6,11],[7,11],[8,11],[9,11],[9,11],[9,11],[9,11]]
),comida([])),
P = 50.
```

Importante: El objetivo es intentar alcanzar la posición de la primera comida de la lista, realizar el crecimiento del worm y continuar jugando con el resto de las comidas. Para

los movimientos debería emplear intento/4, evitando usar en este predicado avanzar/3, seguro/1 y movimiento/4.

Anexo

En el archivo **anexo.pl** está disponible el predicado grafica/1, el cual muestra una configuración del juego similar a la Figura 1 (a excepción de las puertas que no se visualizan). Es únicamente a los fines de ayudar al seguimiento del worm en el plano, no realiza ninguna validación de las posiciones.
`grafica(W)`: W unifica con una función worm. Muestra un plano (según las dimensiones indicadas en 3.a), la posición del worm (obtenidas de W) y los obstáculos del juego (de acuerdo a 3.b).

En el anexo también se incluyen las siguientes reglas para los predicados 3.a – 3.e:

```
plano(11,11).
obstaculos([[2,2],[2,3],[3,2],[3,3],[3,8],[3,9],[4,8],[4,9],[7,4],
[7,5],[7,6],[9,4],[9,5],[9,6]]).
puerta([1,6],[11,6],a).
puerta([11,6],[1,6],b).
puerta([6,11],[6,1],d).
puerta([6,1],[6,11],i).
puntaje(10).
wormCrecimiento(3).
```