Trabajo Práctico N°4: Programación Lógica

Paradigmas de Lenguajes de Programación
2do cuatrimestre 2020

Fecha de entrega: 26 de noviembre de 2020 (antes de las 17 horas)

Introducción

En este trabajo modelaremos ejércitos del popular videojuego Age of Empires II[1]. El juego consiste en crear una civilización que incluye construir edificios, entrenar guerreros, desarrollar tecnologías, etc. Un aspecto central en el juego son los enfrentamientos bélicos contra las civilizaciones enemigas que pueden ser controladas por otros jugadores o por la computadora. Para simplificar el modelo, consideraremos que los ejércitos se dividen en batallones y que los enfrentamientos se producen de a un batallón a la vez. Representaremos entonces a los ejércitos como listas de batallones y a cada batallón como una tupla `(U,C)` donde `U` es el tipo de unidad y `C` es la cantidad de unidades del batallón. Por ejemplo, `(lancero,3)` representa un batallón de 3 lanceros y `[(jinete,2),(jinete,2)]` representa un ejército con dos batallones de 2 jinetes cada uno. Se sabe por contexto de uso que **todo batallón tiene al menos una unidad** y **todo ejército tiene al menos un batallón**

Las unidades se dividen entre unidades militares y aldeanos. Los aldeanos recolectan recursos necesarios para construir edificios y entrenar unidades militares. Las unidades militares pueden ser de distinto tipo. Cada unidad militar tiene un costo de entrenamiento y requiere tener construido un edificio particular para completar su entrenamiento. Llamaremos `pueblo` al conjunto de aldeanos, edificios, y ejército de una civilización. Representaremos al pueblo con un número natural correspondiente a la cantidad de aldeanos, una lista de edificios construidos y un ejército. Por ejemplo, el pueblo representado por `5`, la lista `[establo]` y el ejército `[(jinete,2)]` es un pueblo que tiene 5 aldeanos, un establo y un ejército con un batallón de dos jinetes.

Se proveen los predicados `unidad` (que indica cuáles son las unidades militares disponibles), `edificio` (que indica cuáles son los edificios que se pueden construir), `entrena` (que indica en qué edificio se entrena cada unidad) y `costo` (que indica el costo de cada unidad y de cada edificio).

[1] `https://es.wikipedia.org/wiki/Age_of_Empires_II:_The_Age_of_Kings`

Predicados pedidos

Ejercicio 1. Extender el predicado `costo(+L, -C)` para que funcione cuando se instancia el primer parámetro con una lista. La lista puede ser tanto de batallones como de edificios. En ambos casos el resultado debe ser el costo total de recursos necesarios para entrenar a todas las unidades o construir todos los edificios de la lista. Por ejemplo:

```
?- costo([(lancero,3),(arquero,2),(jinete,1)],C).
  C = 540.

?- costo([establo, cuartel],C).
  C = 700.
```

Ejercicio 2.

a. Definir el predicado `ejercito(-E)` que instancia en `E` **todos** los posibles ejércitos que se pueden formar (es decir, listas no vacías de batallones).

b. Analizar la reversibilidad del parámetro `E`.

Ejercicio 3.

a. Definir el predicado `edificiosNecesarios(+Ej, -Ed)` que dado un ejército instancia en el segundo parámetro la lista de edificios necesarios para poder entrenar a todas las unidades en él (no importa el orden). Por ejemplo:

```
?- edificiosNecesarios([(arquero,3),(jinete,3),(guerrillero,1)],Ed).
  Ed = [arqueria,establo].
```

b. Analizar la reversibilidad de cada parámetro del predicado. Nota: analizar cada uno por separado y no todas las combinaciones posibles entre ambos.

c. Si el predicado anterior no es reversible en el parámetro `Ej`, definir el predicado `edificiosNecesarios2(-Ej, -Ed)` que instancia un ejército y la lista de edificios necesarios para poder entrenar a todas las unidades en él. Indicar si este nuevo predicado es o no reversible en `Ej`.

Ejercicio 4. Para determinar el resultado de un enfrentamiento entre batallones debemos conocer las ventajas y desventajas entre cada par de unidades. Para ello definimos el *índice de superioridad* de una unidad sobre otra, que representa qué tan fuerte es la primera respecto a la segunda. Un índice de superioridad mayor a 1 implica que la primera unidad es más fuerte mientras que un índice menor a 1 implica que la segunda lo es.

Se cuenta con el predicado `ids(+A, +B, -I)` que calcula el índice de superioridad de una unidad sobre otra instanciando el tercer parámetro con dicho valor. Sin embargo, este solo funciona en algunos casos particulares.

a. Modificar la implementación de este predicado para que:

   a) funcione cuando los primeros dos argumentos corresponden a la misma unidad. En este caso se debe instanciar el tercer parámetro en 1.

b) funcione cuando el par de los primeros dos argumentos se corresponde a uno de los ya contemplados pero en el orden inverso. En este caso se debe instanciar el tercer parámetro con el inverso multiplicativo del caso contemplado. Por ejemplo, el índice del jinete sobre el lancero debe ser 1 sobre el índice del lancero sobre el jinete.

c) no se cuelgue ni genere soluciones repetidas.

Está permitido modificar las cláusulas ya definidas, pero es importante que siga funcionando en los casos para los que ya funcionaba, además de los nuevos.

b. Analizar la reversibilidad del predicado actualizado en el parámetro `I`.

Ejercicio 5. Se encuentran implementados los predicados `ids(+A,+B,-I)`, que instancia en `I` el índice de superioridad para los batallones `A` y `B`, y `gana(+A,+B)` que es verdadero cuando el ejército (o batallón) `A` le gana al ejército (o batallón) `B`. Éste último está implementado para emular el siguiente comportamiento: cuando se enfrentan dos ejércitos combaten primero los dos batallones que ocupan la primera posición en sus respectivas listas. El batallón que pierde la contienda es eliminado y el batallón que le sigue en su lista pasa a tomar su lugar, mientras que el batallón que obtiene la victoria se mantiene intacto y continúa luchando hasta que pierda un enfrentamiento contra un batallón enemigo. La batalla finaliza cuando alguno de los dos ejércitos pierde todos sus batallones. En caso de empate se declara como ganador al primero (esto permite que entre dos elementos cada uno le gane al otro pero lo consideraremos un caso borde irrelevante).

a. Definir el predicado `ganaA(?A, +B, ?N)` que funciona como `gana` pero que además permite instanciar el primer parámetro con un batallón o un ejército que le gane al segundo. También posee un tercer parámetro correspondiente a la cantidad de unidades en el batallón o ejército generado. Si este tercer parámetro viene instanciado, el batallón o ejército generado debe tener ese tamaño. Si no, se deben generar todos los posibles batallones o ejércitos que derroten al enemigo usando a lo sumo la misma cantidad de unidades. Por ejemplo:

```
?- ganaA(A,[(arquero,2)],2).
A = [(lancero, 1),  (guerrillero, 1)] ;
A = [(arquero, 1),  (guerrillero, 1)] ;
A = [(jinete, 1),  (guerrillero, 1)] ;
A = [(guerrillero, 1),  (lancero, 1)] ;
A = [(guerrillero, 1),  (arquero, 1)] ;
A = [(guerrillero, 1),  (jinete, 1)] ;
A = [(guerrillero, 1),  (guerrillero, 1)] ;
A = [(arquero, 2)] ;
A = [(jinete, 2)] ;
A = [(guerrillero, 2)] ;
false.

?- ganaA(A,[(arquero,2)],_).
A = [(guerrillero, 1)] ;
A = [(lancero, 1),  (guerrillero, 1)] ;
A = [(arquero, 1),  (guerrillero, 1)] ;
A = [(jinete, 1),  (guerrillero, 1)] ;
A = [(guerrillero, 1),  (lancero, 1)] ;
A = [(guerrillero, 1),  (arquero, 1)] ;
A = [(guerrillero, 1),  (jinete, 1)] ;
```

```
A = [(guerrillero, 1),  (guerrillero, 1)] ;
A = [(arquero, 2)] ;
A = [(jinete, 2)] ;
A = [(guerrillero, 2)] ;
false.
```

b. ¿Utilizaron el predicado `ejercito` como auxiliar para definir `ganaA`? Explicar por qué o por qué no.

Ejercicio 6. Como se mencionó anteriormente, un pueblo está conformado por los aldeanos, los edificios y el ejército entrenado. Para que un pueblo sea válido la cantidad de aldeanos debe ser suficiente como para obtener los recursos necesarios para construir todos sus edificios y entrenar a todas sus unidades militares. Además, debe contar entre sus edificios todos los necesarios para entrenar a todas las unidades de su ejército.

Teniendo en cuenta que cada aldeano provee 50 unidades de recursos, definir el predicado `puebloPara(+En, ?A, -Ed, -Ej)` que dado un ejército enemigo instancia un pueblo (la cantidad de aldeanos, la lista de edificios construidos y el ejército entrenado) cuyo ejército puede derrotarlo. Solo se admiten pueblos cuyo ejército tenga a lo sumo tantas unidades como el ejército enemigo. Si `A` está instanciado, solo instancia los edificios y el ejército de forma que ambos sean válidos para la cantidad de aldeanos disponible. *Sugerencia*: usar la función *ceiling*.

Ejercicio 7. Definir el predicado `puebloOptimoPara(+En, ?A, -Ed, -Ej)` que dado un ejército enemigo instancia un pueblo óptimo (que utiliza la menor cantidad de aldeanos posible) cuyo ejército puede derrotarlo. Si hay más de uno, debe generarlos todos. **No es necesario optimizar la cantidad de recursos, solamente la de aldeanos**. Si `A` está instanciado, solo tiene éxito si la cantidad indicada es la óptima.

Pautas de Entrega

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado.

Se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `tp4.pl`.

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos

- utilización de unificación, backtracking, generate and test y reversibilidad de los predicados.
- **Importante**: salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas ni colgarse luego de devolver la última solución. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía* → *Programación Lógica*).

Se recomienda que utilicen los predicados ISO y los de SWI-Prolog ya disponibles, siempre que sea posible. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Útil* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.