Trabajo Práctico N°3: Programación Lógica

Paradigmas de Lenguajes de Programación
2do cuatrimestre 2021

Fecha de entrega: 23 de noviembre de 2021 (antes de las 17 horas)

Introducción

En este trabajo modelaremos equipos del popular videojuego League of Legends[1]. El juego consiste en elegir a un campeón, formar un equipo, y pelear contra otro equipo hasta destruir su base. Vamos a simplificar el modelo de manera tal que el comportamiento sea el siguiente: cada equipo estará representado por una lista de campeones, en donde el orden determina quién atacará primero. Cuando dos equipos se enfrentan, ataca el primer campeón del equipo atacante, lo que tendrá consecuencias en el equipo defensor. Luego, el campeón que atacó pasará al final de la lista, y ahora es el turno del otro equipo.

Los campeones están representados por el predicado `campeon/1` cuyo argumento es una tripla correspondiente al nombre, la vida (hp) y el daño de ataque (ad). Por ejemplo:

```
campeon((morgana,200,30)).
campeon((chogat,400,50)).
campeon((soraka,300,10)).
campeon((akali,200,150)).
campeon((teemo,190,40)).
```

Cada campeón está asociado a una categoría que indica su rol en las peleas. Las categorías son: mago, tanque, soporte y asesino. Si bien no son la representación fiel del juego original, intentan mantener algunas características principales. Los magos pueden lanzar hechizos que disminuyen el daño de ataque de los oponentes. Los tanques se caracterizan por tener mucha vida, y atacar a todos los rivales al mismo tiempo, aunque no tienen mucho daño de ataque. Los soportes se encargan de mantener a sus aliados con vida y de aumentar su daño de ataque. Y los asesinos tienen mucho daño de ataque, pero solo pueden atacar a un campeón a la vez. La categoría de cada campeón se asigna a través del predicado `tipo/2`:

```
tipo(morgana,mago).
tipo(chogat,tanque).
tipo(soraka,soporte).
tipo(akali,asesino).
tipo(teemo,mago).
```

El objetivo de este trabajo será modelar en Prolog las peleas entre dos equipos.

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

Predicados pedidos

Ejercicio 1.

a. Definir el predicado `equipoInfinito(-E)` que instancia en `E` todos los posibles equipos (pueden tener campeones repetidos). Notar que el orden de los campeones importa (para las peleas) por lo que consideraremos equipos distintos a aquellos que tienen los mismos campeones pero en distinto orden. Consideramos que los equipos no pueden ser vacíos.

b. Analizar la reversibilidad del parámetro `E`.

Ejercicio 2.

a. Definir el predicado `equipoValido(-E)` que instancia en `E` todos los posibles equipos válidos de cantidad menor o igual a 4 de campeones. El equipo es válido si no tiene campeones repetidos, ni tiene campeones del mismo tipo (por ejemplo, `teemo` nunca puede estar en el mismo equipo que `morgana`), ni es vacío.

b. Analizar la reversibilidad del parámetro `E`.

> **IMPORTANTE:** de ahora en adelante sólo consideraremos a los equipos "válidos". Además, en los próximos ejercicios, los equipos que estén instanciados para pelear no comparten campeones.

Ejercicio 3. Definir el predicado `stepPelea(+E1,+E2,-E1F,-E2F)` que instancia en `E1F` y en `E2F`, cómo quedan los equipos `E1` y `E2` luego de un paso de pelea. Como mencionamos anteriormente, el campeón que ataca será el primero en la lista `E1`. Los ataques de cada categoría de campeón son los siguientes:

- **mago:** baja el *ad* de todos los campeones del equipo rival
- **tanque:** baja el *hp* de todos los campeones del equipo rival
- **soporte:** sube el *ad* y el *hp* de todos los campeones de su mismo equipo
- **asesino:** baja el *hp* del primer enemigo del equipo rival

Las cantidades de *ad* y *hp* que se modifican dependen del *ad* del campeón que ataca. Una vez finalizado el ataque, este campeón debe pasar al final de la lista de su equipo. Si algún campeón del equipo rival queda con *hp* menor o igual a 0, éste muere y es eliminado de la lista. Si algún campeón queda con *ad* igual a 0 (no puede ser menor), no pasa nada, simplemente sus ataques no tendrán efecto hasta que algún campeón de tipo soporte lo aumente.

Ejercicio 4. Definir el predicado `pelea(+E1,+E2,+C,-G)` que instancia en `G` al equipo que ganó luego de una cantidad `C` de pasos de pelea. Un equipo gana si en `C` pasos queda con más campeones que el equipo rival (la pelea puede terminar antes si algún equipo se queda sin campeones). Si luego de `C` pasos ambos equipos conservan la misma cantidad de campeones, será considerado un empate y se debe instanciar en `G` el equipo vacío. Notar que los equipos ganadores de las peleas con diferentes cantidades de pasos pueden ser distintos.

Ejercicio 5. Definir el predicado `gana(?E1,+E2)` que es verdadero cuando el equipo `E1` le gana al equipo `E2` en 10 o menos pasos. Si `E1` no está instanciado, deberá instanciarse con todos los equipos válidos que tengan la misma cantidad de campeones que `E2`, y que no contengan los mismos campeones.

Ejercicio 6. Una vez que termina una pelea, se pueden dar honores al mejor jugador. Definir el predicado `honor(+E1,+E2,-C)` que instancia en `C` el nombre del campeón del equipo ganador cuya diferencia entre el *hp* inicial y el final, luego de una pelea de 10 o menos pasos, es la menor. Notar que puede haber más de un campeón cuya diferencia sea la menor, en ese caso se debe instanciar a `C` con todos ellos. El predicado no está definido para el caso de empate en la pelea.

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 `tp3.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, y acompañado de un conjunto de tests que muestren que los predicados funcionan. 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.