Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo Práctico 2

Introducción a la Inteligencia Artificial

SISTEMA EXPERTO EN DotA

Biasín Franco, Rodríguez Jeremías, Meli Sebastián

27 de marzo de 2017

Índice

1. Introducción

DotA (Defense of the Ancients) es un videojuego del género estrategia de acción en tiempo real. Las partidas se juegan de a 10 jugadores divididos en dos equipos de 5, donde cada jugador debe elegir un personaje (héroe) de entre los 112 [1] actualmente disponibles.

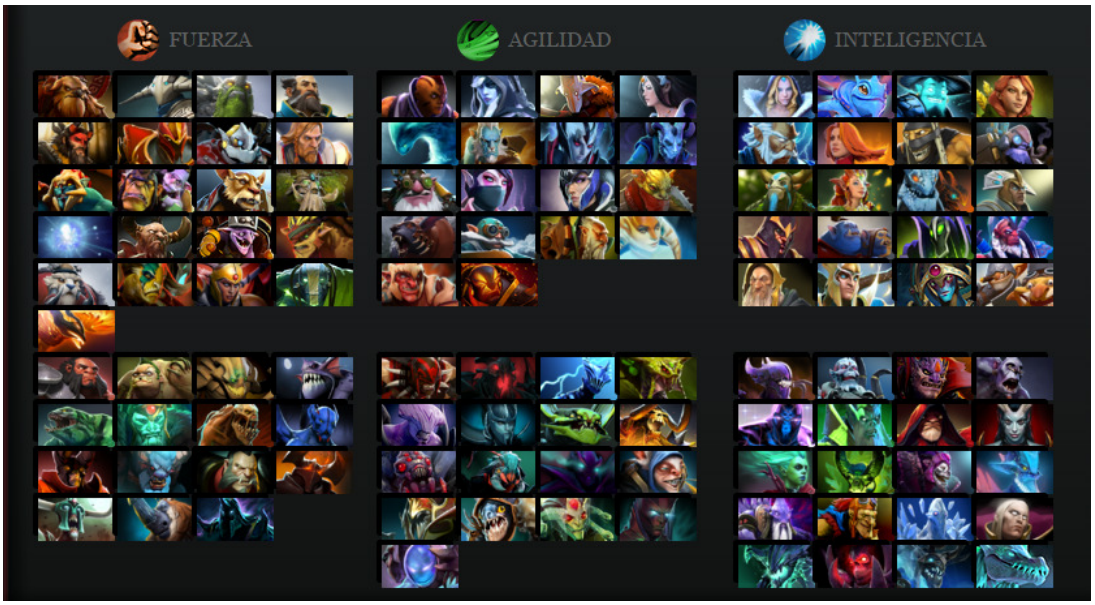
Los héroes son poderosas unidades controladas por el jugador con habilidades especiales únicas. Aunque muchos héroes tienen roles similares a otros, cada uno otorga diferentes ventajas y limitaciones al equipo.

Aprovechando que los héroes se clasifican en distintas categorías, y se pueden agrupar de acuerdo a sus roles en el juego, decidimos hacer un pequeño sistema experto. El objetivo es que el sistema busque y verifique una hipótesis, generando las preguntas necesarias, y al final informe «el héroe buscado es...».

2. Clasificación

En esta sección describiremos las clasificaciones de los héroes y las características que usaremos para identificarlos y generar las preguntas que nos permitirán adivinar cuál es el héroe buscado.

Una primera clasificación bastante general, nos permite dividir a todos los héroes en 3 grandes grupos disjuntos, de acuerdo a su **atributo primario**.



- Los héroes con atributo primario «inteligencia» tienen mucho mana, y muchos skills (habilidades).
- Los héroes con atributo primario «agilidad» tienen una alta velocidad de ataque y mucha armadura.
- Los héroes con atributo primario «fuerza» son muy resistentes y se recuperan rápidamente.

Utilizaremos esta clasificación para comenzar a adivinar el héroe buscado, empezando por averiguar su atributo primario. Haremos preguntas relativas a estas características que hemos dado.

[1] Más info en https://es.wikipedia.org/wiki/Defense_of_the_Ancients

En Prolog, formalizamos el concepto de Héroe y de sus atributos primarios de la siguiente forma:

```
% Nuestro sistema clasifica héroes del juego DotA.
dotaHero(X)  :- satisface(es_un_héroe_de_dota,X),satisface(tiene_skills,X).

% Subcategorias de héroes segun su atributo principal.
inteligencia(X) :- dotaHero(X), satisface(tiene_mucho_mana,X) , satisface(tiene_muchos_skills,X).
agilidad(X)     :- dotaHero(X), satisface(ataca_muy_rapido,X) , satisface(tiene_mucha_armadura,X).
fuerza(X)       :- dotaHero(X), satisface(resistente,X)       , satisface(se_recupera_rapidamente,X)
```

El criterio fue hacer algo análogo al ejemplo brindado en el enunciado. En vez de animales, utilizados héroes de DotA. Seguimos la misma estructura que el ejemplo del enunciado para hacer estos últimos predicados.

Además de su atributo primario, un héroe puede tener una o varias de las siguientes características, de acuerdo a su rol en el juego:

- asistente: no necesita de oro para ser útiles.
- devastador: puede infligir mucho daño en poco tiempo.
- rango: puede atacar a distancia.
- iniciador: tiene habilidades para iniciar la batalla.
- escapista: puede desplazarse grandes distancias en poco tiempo.
- incapacitador: tiene habilidades para inmovilizar a sus oponentes.

Combinando el atributo primario con estas características, podemos identificar los distintos héroes de nuestro juego.

3. Individuos

Los individuos de nuestra base de conocimientos, (análogamente a como son el tigre, chita, etc en el ejemplo de los animales) son algunos héroes que elegimos como ejemplo. En la última sección vemos una extensión para agregar nuevos héroes a la KB.

En Prolog lo escribimos de la siguiente forma:

```
% Distintos héroes (individuos) de nuestra KB:
eartshaker(X)   :- fuerza(X),satisface(iniciador,X),satisface(incapacitador,X),satis (...)
omniknight(X)   :- fuerza(X),satisface(devastador,X),satisface(asistente,X).
timbersaw(X)    :- fuerza(X),satisface(devastador,X),satisface(escapista,X).
juggernaut(X)   :- agilidad(X),satisface(devastador,X),satisface(escapista,X).
void(X)         :- agilidad(X),satisface(iniciador,X),satisface(escapista,X),satisfa (...)
venge(X)        :- agilidad(X),satisface(rango,X),satisface(asistente,X),satisface(i (...)
mirana(X)       :- agilidad(X),satisface(incapacitador,X),satisface(escapista,X),sati(...)
lina(X)         :- inteligencia(X),satisface(incapacitador,X),satisface(devastador,X)(...)
rubick(X)       :- inteligencia(X),satisface(incapacitador,X),satisface(rango,X),sati(...)
ogre(X)         :- inteligencia(X),satisface(incapacitador,X),satisface(devastador,X)(...)
```

Las líneas no entran completas pero la estructura general es esa, análogamente al ejemplo del enunciado.

4. Funcionamiento

Ahora comentaremos brevemente el funcionamiento del sistema en Prolog (con nuestros conocimientos precarios de este lenguaje de programación la explicación puede no ser muy técnica).

```
%%%% Predicado hero
%%%% el predicado principal, se encarga de adivinar qué personaje está pensando el usuario.

hero :- adivina(Hero,_), write('El héroe es: '), write(Hero), nl, borraResp.

%%%% Predicados adivina
%%%% auxiliares del predicado hero

adivina(omniknight,X)  :- omniknight(X).
adivina(earthshaker,X) :- eartshaker(X).
adivina(timbersaw,X)   :- timbersaw(X).
adivina(juggernaut,X)  :- juggernaut(X).
adivina(void,X)        :- void(X).
adivina(mirana,X)      :- mirana(X).
adivina(lina,X)        :- lina(X).
adivina(rubick,X)      :- rubick(X).
adivina(ogre,X)        :- ogre(X).
adivina(venge,X)       :- venge(X).
adivina(desconocido,X).
```

El predicado principal, llamado hero/0, es al que debemos invocar para comenzar el programa. Al ser invocado, lo que se intentará es determinar si es true o false. Para determinar si es true o false, se intenta deducir si es cierto el lado derecho de la única regla que lo define.

Como tenemos una varibale Hero en el predicado adivina(Hero,_), se iterará sobre todos los valores que puede asumir de acuerdo a la definición del predicado adivina.

El predicado adivina tiene una línea por cada individuo de nuestra KB. Es decir, lo que haremos será intentar probar adivina(Hero,X) y ver si vale para algún héroe de nuestra KB. En caso de ser afirmativo, escribiremos el nombre, ingresaremos una nueva línea y finalmente borraremos todas las respuestas a preguntas hechas para poder iniciar nuevamente el programa.

Por lo tanto, lo que el sistema hace es ir probando (en el orden en que están escritos los predicados adivina) cada uno de los predicados que están a la derecha de un «adivina». Si logra demostrar que alguno es cierto, lo expone como respuesta.

Intentará probar si el héroe es omniknight (e.g. omniknight(X) es true), luego si es eartshaker, y así sucesivamente. Por las definiciones dadas en la sección anterior, que definen a cada héroe, esto implicará ir verificando si son true los predicados de la forma «satisface(propiedad,X)».

El orden será de izquierda a derecha. Comenzará verificando que sea un héroe de dota, luego su atributo primario y luego las características extra.

Observar que el último predicado adivina funciona a modo de otherwise, y captura el caso en que no podemos adivinar el héroe.

Ahora veamos la definición de satisface (dada por el enunciado del TP):

```
%%%%  Predicado satisface
%%%%  averigua si un atributo es válido o no, si no lo sabe lo pregunta.
:- dynamic si/1,no/1.
satisface(Atributo,_) :-
```

```
(
si(Atributo) -> true ;
(
no(Atributo) -> fail ;
pregunta(Atributo),satisface(Atributo,_)
)
).


%%%% Predicado pregunta
%%%% consulta al usuario si es un atributo se cumple o no.
pregunta(A) :-
write('¿Tiene el héroe este atributo?: '), write(A), write(' '), read(Resp), nl,
  (
(Resp == s ; Resp == si ; Resp == sí) -> assert(si(A));
assert(no(A))
).
```

Como vemos, se declaran dos predicados dinámicos unarios «Si» y «No». Cuando se intenta inferir si un predicado satisface es true o false, se procesde de la siguiente forma. Si ya le hemos preguntado al usuario, entonces, usamos la respuesta que nos dio antes. Sino, le preguntamos. Para preguntar se utiliza el predicado unario «pregunta», que escribe en pantalla y lee la entrada para ver la respuesta.

Usamos assert para agregar a la KB la respuesta y probablemente reutilizarla.

En líneas generales, ese es el funcionamiento o la lógica del programa: iterar sobre cada posible héroe, en el orden de las líneas que definen «adivina», que elegimos arbitrariamente (del mismo modo que sucedía con los animales en el ejemplo del enunciado). Cuando encontramos una respuesta afirmativa, se la comunicamos la usuario.

Como comentario final a esta explicación muy informal, mostramos el predicado que escribimos para eliminar las aserciones «si» y «no» luego de adivinar una respuesta:

```
%%%% Predicado borraResp
%%%% deshace todas las aserciones si-no agregadas por el predicado pregunta.

borraResp :- retract_yes, retract_no.
retract_yes :- retractall(si(_)).
retract_no  :- retractall(no(_)).
```

5. Ejemplo de Uso

Ejemplo de uso (copiado de la terminal):

```
?- [dota].
true.
?- hero.
¿Tiene el héroe este atributo?: es_un_héroe_de_dota si.
¿Tiene el héroe este atributo?: tiene_skills |: si.
¿Tiene el héroe este atributo?: resistente |: si.
¿Tiene el héroe este atributo?: se_recupera_rapidamente |: si.
¿Tiene el héroe este atributo?: devastador |: si.
¿Tiene el héroe este atributo?: asistente |: si.
```

```
El héroe es: omniknight
true .
```

El código fuente se encuentra en dota.pl.

6. Extensión: Agregando nuevos individuos a la KB

Para esta sección el código fuente se encuentra en dota2.pl.

Escribimos otra versión del sistema que incluye un predicado addNewHero/0 que nos permite agregar nuevos personajes a la KB. Luego de invocarlo, el predicado consiste básicamente en ir haciendo varias preguntas al usuario sobre las características que definen al personaje que desea agregar: nombre, atributo primario, características adicionales.

Para interactuar con el usuario usamos los mismos predicados que ya habíamos usado en «preguntar». Para ir escribiendo las nuevas reglas en el código fuente del mismo archivo que se estaba ejecutando, utilizamos algunas funciones que vimos en clase con Flavio y otras que googleamos.

En particular, la función open[2] en modo append nos permite abrir el archivo dota.pl, y agregar nuevas líneas de texto al final del mismo.

Por este motivo eliminamos el otherwise del predicado «adivina», ya que no permitía al programa utilizar los predicados que se agregaban al final del archivo (lee de arriba hacia abajo, y encuentra el otherwise de «héroe desconocido» antes de leer los que han sido agregados).

Esto ocasiona que el predicado Hero, si se le dan respuestas que no le permiten inferir un resultado, devuelva False y no funcione hasta recargar el archivo. No creemos que sea un problema, porque si no pudo inferir el resultado entonces el usuario podría recurrir a AddNewHero para agregar el nuevo personaje y luego necesariamente deberá recargar el archivo ([dota].) para que los cambios sean tenidos en cuenta.

Un ejemplo de uso:

```
?- addNewHero.
Agregando un nuevo héroe al sistema experto...
No usar mayusculas en el proceso.
Ingrese el nombre del héroe:maiden.
Su atributo primario es inteligencia?|: si.
Desea agregar alguna caracteristica mas del héroe?|: si.
Escriba la caracteristica (solo letras y guiones):|: rango.
Desea agregar alguna caracteristica mas del héroe?|: si.
Escriba la caracteristica (solo letras y guiones):|: asistente.
Desea agregar alguna caracteristica mas del héroe?|: si.
Escriba la caracteristica (solo letras y guiones):|: incapacitador.
Desea agregar alguna caracteristica mas del héroe?|: no.
true.
```

Luego de esto, las últimas líneas del código fuente pasaron a ser:

```
150
151  %%%%%%%%%%%%%%% NUEVOS PERSONAJES AGREGADOS POR EL USUARIO %%%%%%%%%%%%%%%
152  % Aquí comenzarán a agregarse predicados agregados por el usuario, para ampliar la KB.
153  % Se verán warnings ocasionados por el desorden de las cláusulas adivina.
154
155  adivina(maiden,X) :- maiden(X).
156  maiden(X):-inteligencia(X),satisface(rango,X),satisface(asistente,X),satisface(incapacitador,X).
157
```

[2] http://www.swi-prolog.org/pldoc/man?predicate=open/4, no pudimos encontrar una función parecida que nos deje escribir en una línea particular del archivo.