SISTEMAS EXPERTOS
DISEÑO DE VIDEOJUEGOS

Manuel Palomo Duarte
José Tomás Tocino García

Junio de 2011

ÍNDICE

ÍNDICE

¿QUÉ ES UN SISTEMA EXPERTO?

- **Sistema experto:** mecanismo que **simula** el conocimiento de un experto humano en una materia determinada.
- Se usan con éxito en muchas ramas de la ciencia: medicina, ingeniería, etc.
- Existen varios tipos:
  - Basados en **reglas**. Son los que estudiaremos.
  - Basados en **casos**.
  - Basados en **redes bayesianas**.

ÍNDICE

COMPONENTES PRINCIPALES

HECHOS

Información sobre el entorno que el sistema lee y utiliza para tomar decisiones.

REGLAS

Condiciones que el sistema evalúa a partir de los hechos presentes para generar nuevo conocimiento.

MOTOR DE INFERENCIA

Se encarga de decidir qué reglas pueden activarse según los hechos presentes.

COMPONENTES SECUNDARIOS

LISTA DE ACTIVACIÓN (DEL INGLÉS *agenda*)

Contiene las reglas cuyas condiciones se han cumplido y son candidatas a dispararse.

ESTRATEGIAS DE RESOLUCIÓN DE CONFLICTOS

Habrá ocasiones en las que varias reglas podrán ser candidatas, hay que decidir qué reglas disparar en cada caso.

ÍNDICE

FUNCIONAMIENTO

1. Se leen los hechos.
2. Se comprueba qué reglas cumplen las condiciones.
3. Se añaden las reglas candidatas a la agenda.
4. Se lanzan las reglas de la agenda, generando y/o borrando hechos como resultado de su ejecución.
5. Vuelta al principio.

ÍNDICE

CLIPS

Usaremos **CLIPS** como sistema para el desarrollo y ejecución de sistemas expertos basados en reglas.

- Es un sistema open source, creado por la NASA y mantenido por uno de sus fundadores.
- Existen muchos wrappers y derivados en otros lenguajes para poder interactuar con Clips.
- Más información en `http://clipsrules.sourceforge.net`.

HECHOS EN CLIPS

Un **hecho** en clips tiene la siguiente forma:
`(<relación> <campos_de_información>)`

Por ejemplo:
`(persona "Pepe")`

Los hechos se *afirman* con `assert`:
`(assert (persona "Pepe"))`

Y se retractan con `retract`.
`(retract <num_hecho>)`

Se puede utilizar `(facts)` para conocer los hechos y sus números asignados.

HECHOS EN CLIPS

Los **hechos iniciales** se indican con `deffacts`:

```
(deffacts
 (assert (persona "Pepe" 15))
 (assert (persona "Juan" 18))
 (assert (trabajo "Pepe" "Docente"))
 (assert (trabajo "Juan" "Estudiante"))
)
```

REGLAS EN CLIPS

Las **reglas** en CLIPS tienen dos partes:

1. **Condiciones:** serie de hechos y patrones que deben cumplirse para que la regla se active.
2. **Acciones:** si las condiciones es cumplen, estas acciones se lanzarán, normalmente generando nuevos hechos.

Siguen esta sintaxis:

```
(defrule <nombre_regla>
  <condiciones>
    =>
  <acciones>
)
```

REGLAS EN CLIPS

Por ejemplo:

```
(defrule apagar_fuego
  (hay_emergencia fuego)
   =>
  (assert (llamar bomberos))
)
```

Podemos declarar la prioridad de una regla con `salience`:

```
(defrule <nombre_regla>
  (declare (salience 50))
   ...
```

REGLAS EN CLIPS

Podemos usar **condiciones genéricas** que valgan para muchos hechos. Por ejemplo, esta regla se ejecutará para todas las personas, guardando el nombre de cada persona en la variable `?n`.

```
(defrule imprimir_persona
  (persona ?n)
    =>
  (printout t "Existe una persona cuyo nombre es "
    ?n crlf)
)
```

REGLAS EN CLIPS

Para hacer **comprobaciones** arbitrarias, usaremos `test`. Lo usaremos en el ejemplo siguiente.

Es posible guardar **referencias a hechos** en las condiciones para trabajar con ellos en las acciones de la regla:

```
(defrule MODULO::jubila1
  (persona ?n ?e)
  ?h <- (trabajo ?n ?t)
  (test (> ?e 65))
    =>
  (retract ?h)
  (assert (jubilado ?n))
)
```

FUNCIONES EN CLIPS

Podemos modularizar las operaciones en **funciones** con `deffunction`. El valor de retorno será el de la última expresión evaluada:

```
(deffunction MAIN::mayor-mas-uno (?a ?b)
  (if (> ?a ?b) then
    (+ ?a 1)
  else
    (+ ?b 1)
  )
)
```

PLANTILLAS EN CLIPS

Es posible estructurar la información de un hecho mediante el uso de **plantillas**.

```
(deftemplate persona
   (slot nombre)
   (slot edad)
   (slot peso)
)
(assert (persona (nombre "Pepe") (edad 27)))
```

Nos permitirá filtrar por campos individuales:

```
; Persona de edad 27, da igual el nombre o el peso
?h <- (persona (edad 27))
```

ÍNDICE

INTRODUCCIÓN

IDEA

Crear un sistema para el **enfrentamiento de dos ejércitos**, cada uno controlado por un **sistema experto** basado en reglas escritas en CLIPS por los propios alumnos.

PLANTEAMIENTO

- Planteamiento basado en Stratego.
- Tenemos un **tablero** de 8x8, y **dos ejércitos** de 16 fichas:
  - Un **rey** (valor 1), al que hay que defender.
  - Ocho peones (valor 2).
  - Dos fichas de valor 3, dos de valor 4 y dos de valor 5.
  - Una ficha todopoderosa de valor 6.
- Por **turnos**, cada ejército mueve una ficha. Las fichas solo pueden moverse una casilla en horizontal o vertical.
- Cuando dos fichas **colisionan**, se muestran sus valores y muere la de menor valor, o ambas si hay empate.

IMPLEMENTACIONES PREVIAS

El sistema ha evolucionado bastante a lo largo de los años:

1. Versión 0.1, **modo texto**. Totañlmente funcional.
2. Versión 0.1.1, se añade un **visor gráfico** para las partidas de texto.
3. Versión 1.0, **La Reconquista**, aplicación gráfica e interactiva.
4. Versión 2.0, **Resistencia en Cádiz 1812**. Reescritura de la versión 1.0, con pruebas automáticas.
5. Versión 2.x, **Gades Siege**. Ampliación del proyecto Resistencia 1812, mejoras gráficas, nuevas reglas, etc.