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

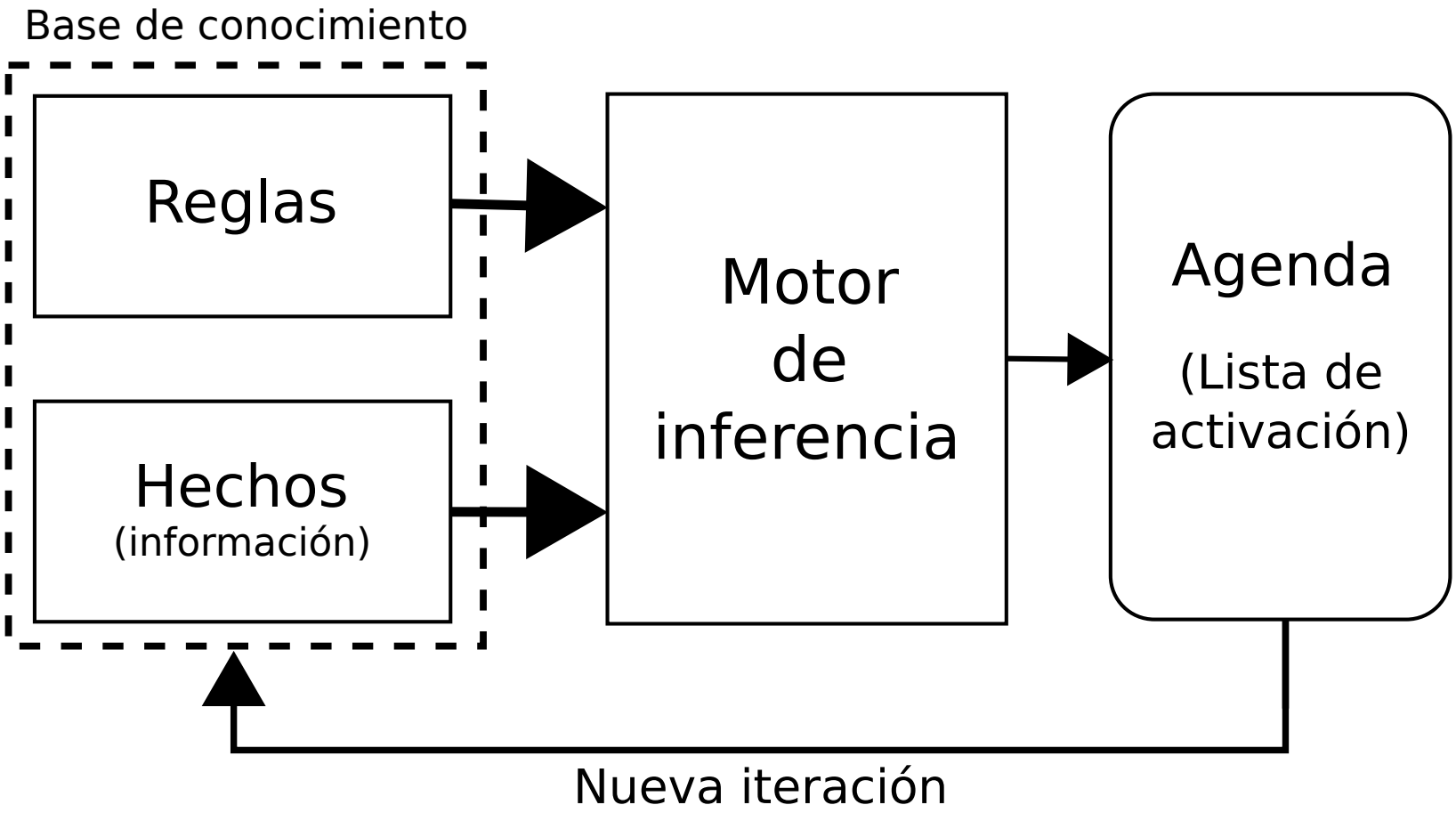
Decide qué regla disparar si hay varias que se cumplen. Puede ser la que más tiempo lleve sin dispararse, la que más veces se halla disparado, una aleatoriamente, etc.

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.

DIAGRAMA DE FUNCIONAMIENTO



ÍNDICE

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 creadores.
- Existen muchos wrappers y derivados en otros lenguajes para poder interactuar con Clips: Jess (CLIPS para Java), PyCLIPS, FuzzyCLIPS, etc.
- Más información en `http://clipsrules.sourceforge.net`.

ÍNDICE

HECHOS EN CLIPS

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

Por ejemplo:
`(persona "Pepe")`

Se añaden hecho al sistema con `assert`:
`(assert (persona "Pepe"))`

Y se eliminan de él 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"))
)
```

ÍNDICE

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 modificando el conjunto de 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` con notación infija.

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))
)
```

ÍNDICE

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)
  )
)
```

ÍNDICE

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

ÍNDICE

INTRODUCCIÓN

OBJETIVO

Aprender el uso de un sistema experto para la inteligencia artificial **de un videojuego**, mediante un videojuego ;-)

IDEA

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

ÍNDICE

PLANTEAMIENTO

- Planteamiento basado en Stratego.
- Tenemos un **tablero** de 8x8, y **dos ejércitos** de 16 fichas:
  - Un **rey** (valor 1), que cuando muere acaba la partida.
  - 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.

HISTORIAL DE VERSIONES

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

1. Versión 0.1, **modo texto**. Totalmente 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.

ÍNDICE

ESTRUCTURA DEL JUEGO: FICHAS

Las **fichas** en Gades Siege se representan con la plantilla `ficha`, que tiene los siguientes *slots*:

- EQUIPO: Puede ser *"A"* (equipo propio) o *"B"* (adversario).
- NUM: Identificador único de la ficha en el equipo.
- PUNTOS: Valor de la ficha.
- POS-X: Posición horizontal.
- POS-Y: Posición vertical.
- DESCUBIERTA: Valdrá 1 si la ficha está descubierta.

ESTRUCTURA DEL JUEGO: GENERAL

- El tablero es de 8x8, información que se guarda en un hecho `(dimension 8)`.
- En cada turno, existe un hecho `(tiempo t)` que indica la cantidad de movimientos restantes posibles.
- Los equipos de cada jugador se definen en dos ficheros: `reglasEquipo.clp` y `equipoEquipo.form`.
- El flujo de ejecución es el siguiente:
  - Se muestra el estado por pantalla.
  - Se realiza un movimiento.
  - Se actualiza el mundo.

ESTRUCTURA DEL JUEGO: FICHERO DE FORMACIÓN

En el **fichero de formación** se guardarán las posiciones iniciales de las 16 fichas de cada equipo.

- Deben guardarse en la ruta `data/teams/formations`
- Su nombre debe seguir la estructura `equipoNombre.form`

Ejemplo de fichero de formación:

```
6:5:5:4:4:3:3:2
2:2:2:2:2:2:2:1
```

Se colocan los valores de cada ficha en dos filas, tal y como aparecerían en el tablero (suponiendo que somos el equipo que juega abajo).

ESTRUCTURA DEL JUEGO: FICHERO DE REGLAS

En el **fichero de reglas** se guardarán las reglas que el sistema experto usará para jugar.

- Deben guardarse en la ruta `data/teams/rules`
- Su nombre debe seguir la estructura `reglasNombre.clp`

En la primera línea del fichero de formación es posible añadir un comentario que se verá luego en la interfaz gráfica, siguiendo la siguiente sintaxis:

```
; DOC: aquí va el comentario
```

De igual modo se pueden añadir comentarios en el resto del fichero usando el punto y coma como indicador.

ESTRUCTURA DEL JUEGO: MOVIMIENTO

En Gades Siege, un **movimiento** se solitica como un hecho mediante una plantilla `mueve`, que tiene los siguientes *slots*:

NUM Identificador de la ficha a mover.

MOV Número de movimiento, según esta relación:

1. avanza X
2. retrocede X
3. avanza Y
4. retrocede Y

TIEMPO Momento del tiempo en el que realizar el movimiento.

Por ejemplo, un movimiento horizontal:

```
(assert (mueve (num ?n) (mov 1) (tiempo ?t)))
```

Las reglas siempre suponen que el jugador empieza en la parte inferior del tablero. si no es así, el sistemas se encarga de traducirlas.

EJEMPLO DE REGLA 1

```
(defrule EQUIPO-A::atacar1
  (declare (salience 30))
  (ficha (equipo "A") (num ?n1) (pos-x ?x1)
         (pos-y ?y1) (puntos ?p1))
  (ficha (equipo "B") (num ?n2) (pos-x ?x2)
         (pos-y ?y2) (puntos ?p2) (descubierta 1))
  (test (and (> ?p1 ?p2) (= ?y1 ?y2) (> ?x1 ?x2)))
  (tiempo ?t)
    =>
  (assert (mueve (num ?n1) (mov 2) (tiempo ?t)))
)
```

EJEMPLO DE REGLA 1, CONDICIONES

`(defrule EQUIPO-A::atacar1` → cabecera de la regla.

`(declare (salience 30))` → declaramos la prioridad de la regla.

`(ficha (equipo "A") (num ?n1) (pos-x ?x1) (pos-y ?y1) (puntos ?p1))` → buscamos una ficha cualquiera de nuestro equipo, guardando su posición y puntos.

`(ficha (equipo "B") (num ?n2) (pos-x ?x2) (pos-y ?y2) (puntos ?p2) (descubierta 1))` → buscamos una ficha adversaria, que esté descubierta.

`(test (and (> ?p1 ?p2) (= ?y1 ?y2) (> ?x1 ?x2)))` → comprobamos que:

- Nuestra ficha sea superior a la otra ficha.
- Ambas fichas estén en la misma columna (igual `pos-y`).
- Nuestra ficha esté más a la derecha que la otra ficha.

`(tiempo ?t)` → guardamos el turno en la variable t.

EJEMPLO DE REGLA 1, ACCIONES

Si todas las condiciones anteriores se cumplen, esto es, se han encontrado fichas que cumplan los criterios indicados, se ejecutan las acciones.

```
(assert (mueve (num ?n1) (mov 2) (tiempo ?t)))
```

Mueve nuestra ficha en la dirección de la ficha adversaria (con intención de capturarla).

EJEMPLO DE REGLA 2

```
(defrule EQUIPO-A::atacar2
  (declare (salience 20))
  (ficha (equipo "A") (num ?n1) (pos-x ?x1)
         (puntos ?p1))
  (ficha (equipo "B") (num ?n2) (pos-x ?x2)
         (puntos ?p2) (descubierta 1))
  (test (and (> ?p1 ?p2) (> ?x1 ?x2)))
  (tiempo ?t)
    =>
  (assert (mueve (num ?n1) (mov 2) (tiempo ?t)))
)
```

EJEMPLO DE REGLA 3

```
(defrule EQUIPO-A::huir1
  (declare (salience 20))
  (ficha (equipo "A") (num ?n1) (pos-x ?x1)
         (pos-y ?y1) (puntos 1))
  (ficha (equipo "B") (pos-x ?x2) (pos-y ?y2)
         (puntos 5))
  (test (and (= ?x1 ?x2) (> ?y1 ?y2)))
  (tiempo ?t)
   =>
  (assert (mueve (num ?n1) (mov 4) (tiempo ?t)))
)
```

EJEMPLO DE REGLA 4

```
(defrule EQUIPO-A::despistar
  (declare (salience 10))
  (ficha (equipo "A") (num 11) (pos-x ?x1)
         (pos-y ?y1))
  (test (< ?y1 4))
  (not (ficha (equipo "A") (pos-x ?x1)
              (pos-y (+ 1 ?y1))))
  (tiempo ?t)
    =>
  (assert (mueve (num 11) (mov 3) (tiempo ?t)))
)
```

¡A TRABAJAR!

- Intentad crear reglas *inteligentes* y valientes.
- Disponéis de prioridades entre 1 (mínima) y 80 (máxima).
- Se pueden usar funciones.
- No podéis modificar los hechos de tiempo, dimensión y fichas.
- Se pueden usar hechos auxiliares si se desea.
- Existen reglas con prioridad 0 que se disparan si no hay ninguna mejor. Éstas hacen oscilar las fichas entre (4,4) y (5,5).
- Más información en `http://gsiege.googlecode.com`