etsinf

Sistemas Inteligentes

Práctica -1

Diseño, Implementación y Evaluación de un SBR

(CLIPS)

Grupos 3B1 – 3B2
2020 / 21

etsinf

Objetivo: ***Diseñar y evaluar (espacialmente)*** un Sistema Basado en Reglas para el problema propuesto.

1) Se proporciona un **Boletín introductorio del entorno CLIPS** que incluye ejemplos de ***problemas resueltos*** de '*Ordenación*' y '*Secuencias ADN*'.

2) Se proporciona en otro boletín el **Problema del 8-puzzle** resuelto. Incluye varias estrategias de búsqueda no informada (***anchura y profundidad***).

3) El **problema propuesto** 'Recogida de paquetes en un edificio' del SBR a desarrollar.

Entorno CLIPS (sofware libre: http://clipsrules.sourceforge.net/)

etsinf

Planificación

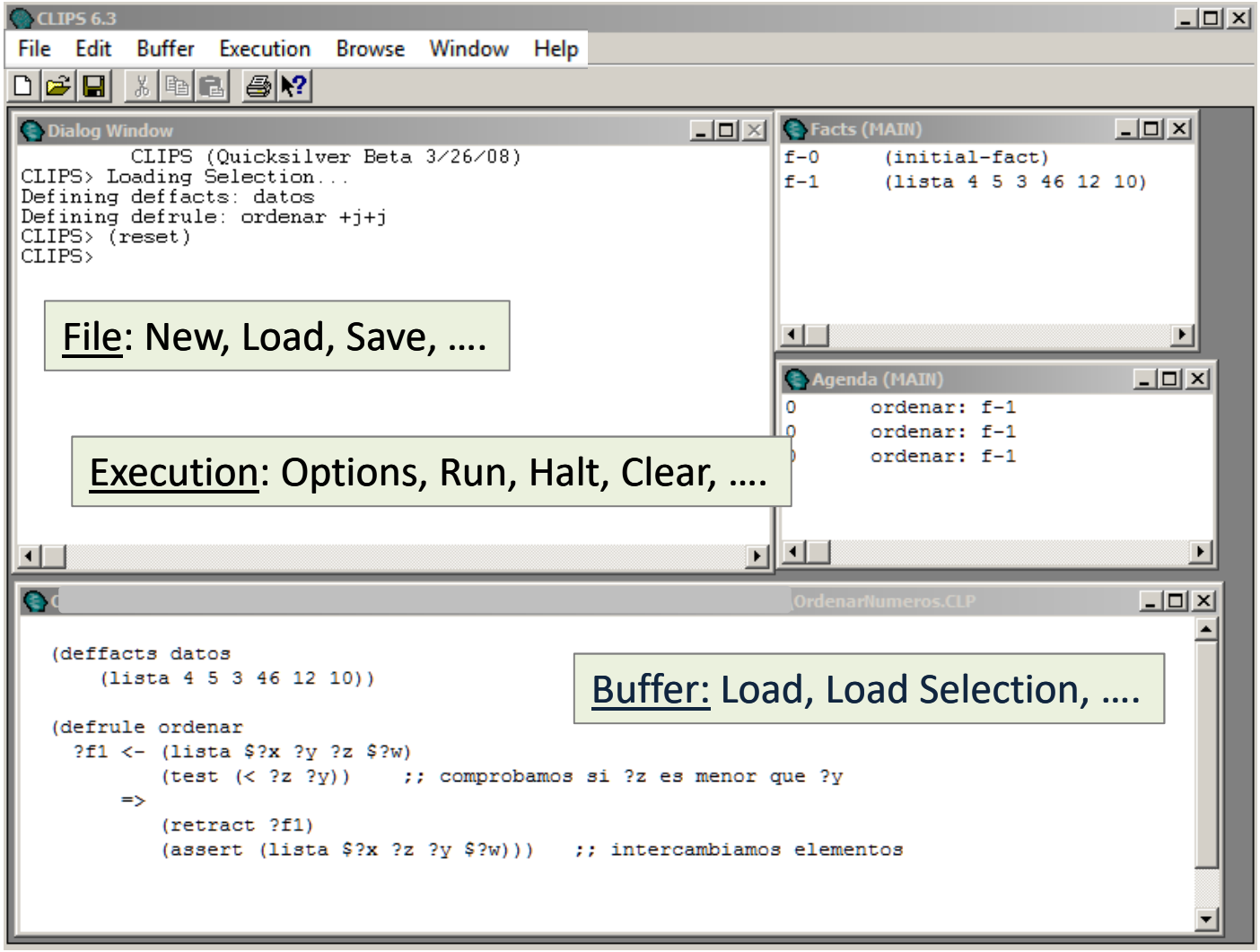
1ª 30-IX	Presentación CLIPS *(ver Manual)* • Boletín Práctica (SIN_BoletinCLIPS.pdf). *Manuales Clips*. Problema Mutaciones, Ordenación. • SBR 8-Puzzle (Anchura y Profundidad): PROBLEMA RESUELTO: PUZZLE • Planteamiento del problema a resolver: PROBLEMA A RESOLVER
7-10 14-10 21-10	Diseño e Implementación SBR (Problema Propuesto)
5ª 28-X	Evaluación INDIVIDUAL

Entorno de CLIPS (BOLETÍN)

- Base de Hechos (hechos)
- Base de Reglas (reglas)
- Motor de Inferencia (control)

Ventanas:

- Dialog
- Facts
- Agenda
- Buffer



(reset)
(run)
(run n)
(clear)

etsinf


```
(deffacts  ejemplo   (……) (……) (…..))

(defglobal     ?*nod-gen*   =   0))

(deffunction nombre  (arg…)  ….)

(defrule regla-1
  ?f    <- ( ……..)
          (test ( …))
=>
          (printout t …)
          (assert (….))
          (retract ?f))
```

Algunas funciones (boletín)

```
(bind   ?x    (+ ?x 1))
(bind   ?x  (read))
(length$    $?x)
(member$   (create$  caja  (+ ?x  1)  4)   $lista-cajas )
(printout  t "SOLUCION EN EL NIVEL " ?n crlf)
```

Patrón (existencia en la BH): (lista ?x $?y ?z)

o patrones negados: (not (caja =(+ ?x 1) ?y))

Nota: en patrón negado incluid =

Predicados:

- (<|=|>   ?n1 ?n2)
- (eq ?s1 ?s2)
- (neq ?s1 ?s2)
- (evenp ?n): TRUE si el número es par
- (floatp ?n): TRUE si es un número en coma flotante
- (integerp ?n): TRUE si es un número entero
- (numberp ?n): TRUE si el argumento es un número
- (oddp ?n): TRUE si el número es impar

Variables globales

```
(defglobal   ?*variable*      =    0)          ;variable mono valuada
(defglobal   ?*lista*    =  (create$))         ;variable multi valuada
```

Declaración de Funciones *(no necesarias)*

(deffunction <function-name> (?arg1 ?arg2 … ?argM [$?argN])

```
(<action1>
……
<action k>)
```

Las principales funciones predefinidas son (ver boletín): while, if then else, break, return

Estrategias (No Informadas: Anchura, Profundidad)

- Elegibles en ***Execution/Options***
- Comandos: ***(set-strategy breadth)***
  (set-strategy depth)

etsinf

Tareas Sesión-1:

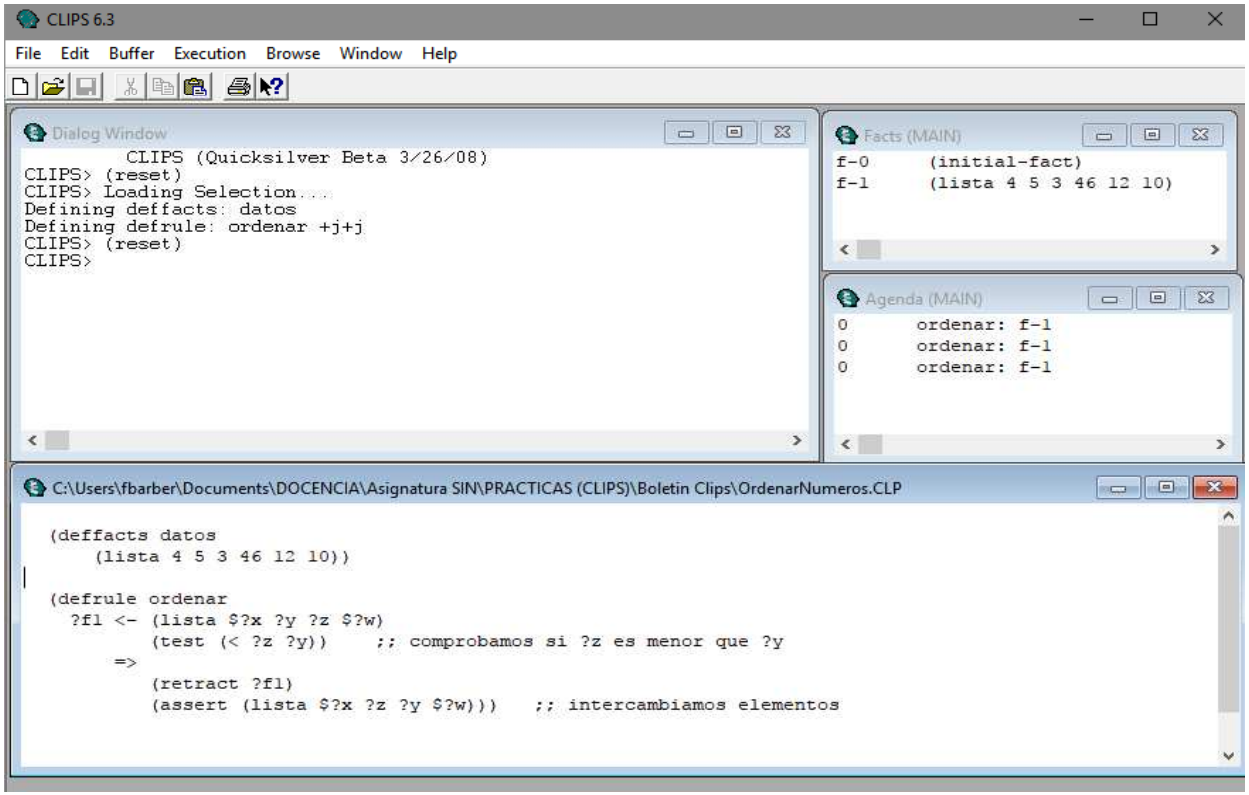
1. Revisar Boletín: Ventanas de CLIPS; Buffer, Reset, Run, etc.
2. Probar ejemplo ordenación números.
3. Probar ejemplo mutaciones ADN.
4. Revisar el problema del puzzle (anchura, profundidad)

Ordenación de una lista de números

```
(deffacts datos
    (lista 4 5 3 46 12 10))
```

(Reset)
(Run)

```
(defrule ordenar
  ?f1 <- (lista $?x ?y ?z $?w)
          (test (< ?z ?y))    ;; comprobamos si ?z es menor que ?y
        =>
          (retract ?f1)
          (assert (lista $?x ?z ?y $?w)))   ;; intercambiamos elementos
```



etsinf

Secuencias de ADN
contar número de mutaciones entre dos secuencias

```
(deffacts datos
    (ADN 1 A A C C T C G A A A)
    (ADN 2 A G G C T A G A A A)
    (mutations 0))

(defrule R_mutation
  ?f1 <-  (ADN ?n1 $?x ?i1 $?y1)
  ?f2 <-  (ADN ?n2 $?x ?i2 $?y2)
  ?f3 <-  (mutations ?m)
          (test (and (neq ?n1 ?n2)(neq ?i1 ?i2)))
  =>
    (retract ?f1 ?f2 ?f3)
    (assert (ADN ?n1 $?y1))
    (assert (ADN ?n2 $?y2))
    (assert (mutations (+ ?m 1))))


(defrule final
  (declare (salience -10))
   (mutations ?m)
 =>  (printout  t "El número de mutaciones es " ?m crlf))
```

(Reset)
(Run)

El problema del 8-puzzle

- Un n-puzzle es una matriz de números desde 0 (vacío) hasta n
- Un estado o configuración es una permutación de estos números. Por ejemplo:

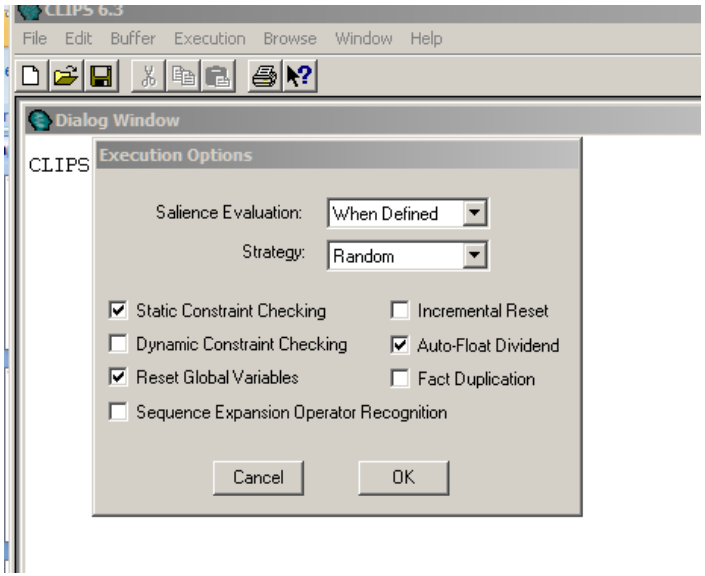
2	0	1
5	3	4
8	6	7

1	2	3
8	0	4
7	6	5

- El objetivo es pasar el n-puzzle de un estado inicial a un estado final (ambos dados) mediante movimientos del número 0.
- El conjunto de movimientos son: arriba, abajo, izquierda, derecha

Tareas:

1. Revisar diseño puzzle *(importante!)*
2. *Función (inicio). Importante!!*
3. Probar configuraciones (función comprobar)
4. Probar búsqueda anchura y profundidad *(solo!)*.
5. *Analizar* los resultados



Puzanpo.CLP *(8-puzzle, anchura y profundidad)*

Representación del estado: Estado, Nivel, Movimiento-Previo , Nodo-padre

(puzzle 8 1 3 7 2 5 4 0 6 nivel 0 movimiento nulo hecho 0)

Movim. Previo (para evitar ciclos)

Nodo padre (para recomponer la senda)

Reglas: 4 movimientos

```
(defrule arriba
  ?f<-(puzzle $?x ?a ?b ?c 0 $?y nivel ?nivel movimiento ?mov hecho ?)
      (profundidad-maxima ?prof)          ;profundidad máxima
      (test (neq ?mov abajo))             ;movimiento inverso
      (test (< ?nivel ?prof))
  =>
  (assert (puzzle $?x 0 ?b ?c ?a $?y nivel (+ ?nivel 1) movimiento arriba hecho ?f))
  (bind ?*nod-gen* (+ ?*nod-gen* 1)))     ;nodos generados. Var. Global: (defglobal ?*nod-gen* = 0))
```

Regla meta: (puzzle 1 2 3 8 0 4 7 6 5 nivel ?nivel movimiento ?mov hecho ?h)

Senda solución: (camino n) ;n: número hecho meta (recupera camino)

Lanzamiento del ejemplo: (inicio) ;determina bh-inicial (deffacts….), estrategia y prof máxima

Comprobar : (comprobar_conf (create$ 2 8 3 1 6 4 7 0 5)) ; Comprueba que sea resoluble

etsinf

Tareas Sesión-1:

1. Revisar Boletín CLISP:

   Ventanas, Editor, Facts, Agenda, Buffer, Reset, Run, etc.

2. Probar ejemplo ordenación números. *Alguna modificación?*

3. Probar ejemplo mutaciones ADN. *Alguna modificación?*

4. Revisar el problema del puzzle *(puzanpo.clp)*

   Anchura, profundidad. Entender código y realizar traza

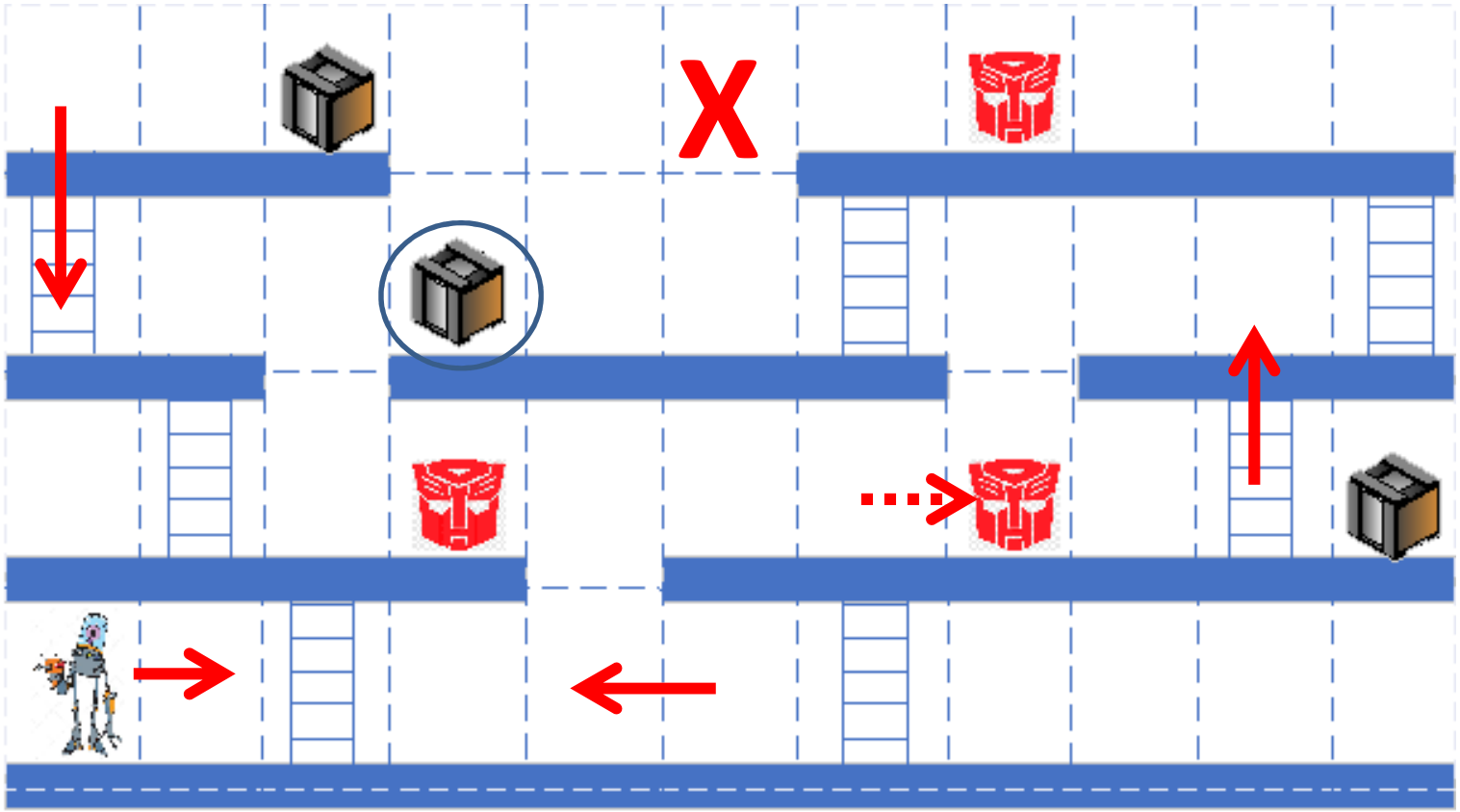
5. Entender Problema Propuesto.

Diseño de un SBR

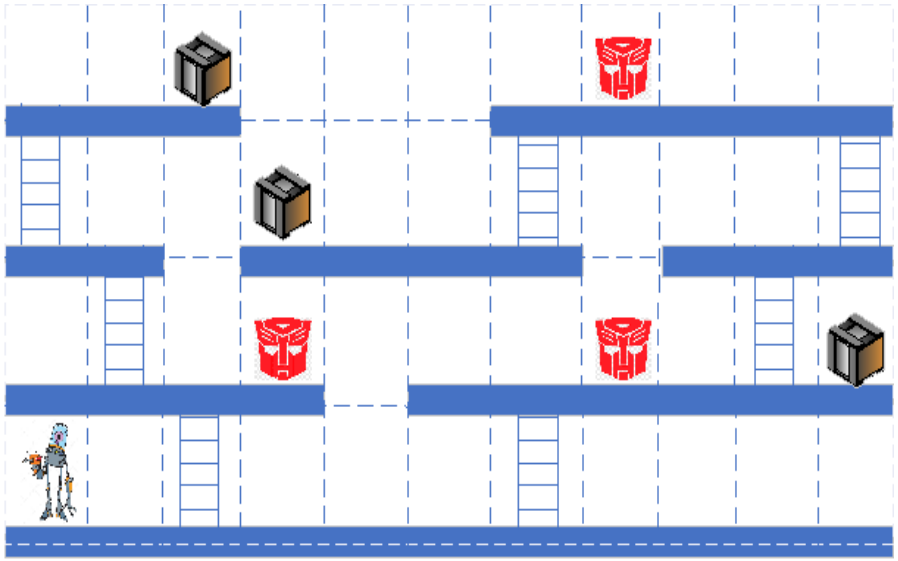
Problema Propuesto

Recogida de paquetes en un edificio

El objetivo es planificar las acciones de un robot para recoger un conjunto de cajas.



etsinf



(deffacts problema …..)

Información Estática:

- Tamaño escenario: (Tamaño 11 4)
- Posición huecos: (hueco 5 2) , ……
- Posición escaleras: (escalera 3 1), ……

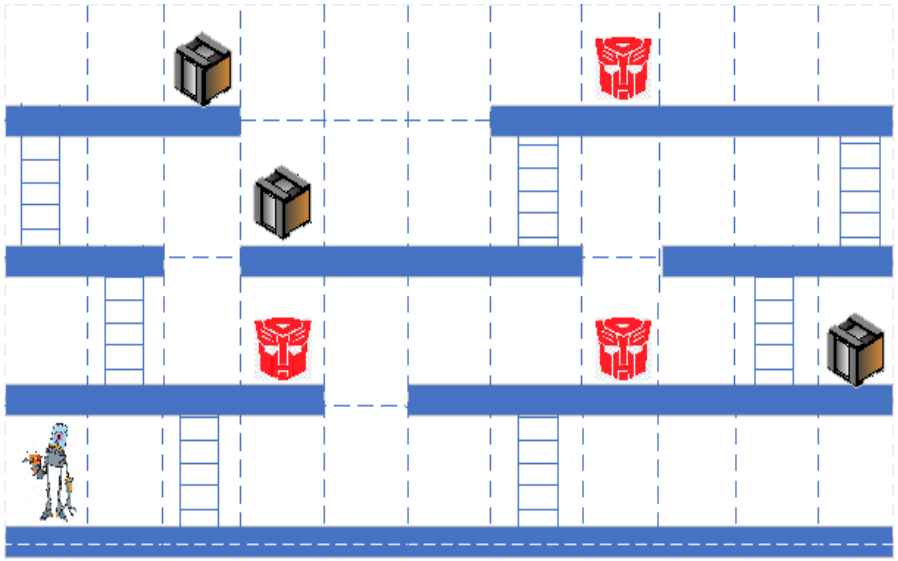
Información dinámica (único patrón)

- Posición del robot
- Posición cajas (sin recoger)
- Posición enemigos (sin inutilizar)
- Número de disparos disponibles
- Nivel nodo

Acciones (defrule …..):

1) Mover robot una casilla a la derecha (si no hueco, ni enemigo). Dentro de límites.
2) Mover el robot una casilla a la izquierda (si no hueco, ni enemigo). Dentro de límites.
3) Subir una planta (si robot bajo de escalera). Dentro de límites.
4) Bajar una planta (si robot arriba de escalera). Dentro de límites.
5) Recoger una caja (si robot en misma casilla que caja).
6) Disparar a enemigo (si robot en casilla adyacente que enemigo). Deben quedar disparos.
7) ***Regla meta: máxima prioridad.***

etsinf



(deffacts problema …..)

Información Estática:

- Tamaño escenario: (Tamaño 11 4)
- Posición huecos: (hueco 5 2) , ……
- Posición escaleras: (escalera 3 1), ……

Información dinámica (único patrón)

- Posición del robot
- Posición cajas (sin recoger)
- Posición enemigos (sin inutilizar)
- Número de disparos disponibles
- Nivel nodo

Ejemplo:

- Dimensión 11x4
- Huecos en casilla (5,2), …. Etc.
- Escalera en la casilla (3,1), (7,1), (2,2),… etc.

- *Robot en casilla (1,1)*
- *Enemigo en la casilla (4,2), (8,2),… etc.*
- *Caja en casilla (4,3), (11,2),… etc.*
- *Disparos disponibles 2*
- *Nivel 0*

Toda la información en hechos iniciales

```
(deffacts  problema
   (tamaño  ….)
   (hueco ……)  ….
   (escalera …….)  ….
   (juego robot   ……. …… ….. …. …..)
)
```

etsinf

Información Estática:

Tamaño escenario: (Tamaño 11 4)
Posición huecos: (hueco 5 2) ……
Posición escaleras: (escalera 3 1) ……

Variables Globales

(defglobal ?*nod-gen* = 0)
(defglobal ?*prof* = 60)

Información dinámica (único patrón):

Fact-n: (Problema Posición …. Cajas ……. Enemigos ….. Disparos ….. Nivel …..)

- Posición del robot: ?x ?y?
- Posición cajas (sin recoger): caja ?cx ?cy **…….**
- Posición enemigos (sin inutilizar): enemigo ?ex ?ey **……..**
- Número de disparos disponibles: ?disp
- Nivel nodo: ?n

Mover_derecha

Dentro de límites?: (+ ?x 1) ?y
No enemigo?: (+ ?x 1) ?y
No hueco?: (+ ?x 1) ?y
Comprobar max-Prof

Información dinámica (único patrón)

- Posición del robot**: (+ ?x 1) ?y**
- Posición cajas (sin recoger)
- Posición enemigos (sin inutilizar)
- Número de disparos disponibles
- Nivel nodo**: (+ ?n 1)**

(bind ?*nod-gen* (+ ?*nod-gen* 1))

etsinf

Información Estática:

Tamaño escenario: (Tamaño 11 4)
Posición huecos: (hueco 5 2) ……
Posición escaleras: (escalera 3 1) ……

Variables Globales

(defglobal ?*nod-gen* = 0)
(defglobal ?*prof* = 60)

Información dinámica (único patrón):

Fact-n: (Problema Posición …. Cajas ……. Enemigos ….. Disparos ….. Nivel …..)

- Posición del robot: ?x ?y?
- Posición cajas (sin recoger): caja ?cx ?cy **…….**
- Posición enemigos (sin inutilizar): enemigo ?ex ?ey **……**
- Número de disparos disponibles: ?disp
- Nivel nodo: ?n

1) **Mover derecha**
2) **Mover izquierda**
3) **Subir una planta**
4) **Bajar una planta**
5) **Recoger una caja**
6) **Disparar a enemigo**
7) ***Regla meta***

Información dinámica (único patrón)

- Posición del robot
- Posición cajas (sin recoger)
- Posición enemigos (sin inutilizar)
- Número de disparos disponibles
- Nivel nodo: **(+ ?n 1)**

(bind ?*nod-gen* (+ ?*nod-gen* 1))

etsinf

IMPORTANTE:

- El sistema ***debe ser válido para cualquier configuración inicial***.
- Toda la información inicial se puede representar directamente en el comando deffacts que contiene los hechos iniciales.
- La representación debe ser generalista (diferentes escenarios: plantas, casillas, escaleras, cajas, enemigos) y permitir **cambios** fácilmente **sin necesidad de modificar las reglas**.

EJECUCION

- Función **Inicio** que pregunte por la profundidad máxima y estrategia ***(ver ejemplo del 8-puzzle).***
- El programa debe **devolver la profundidad** donde se encuentra la solución y el **número de nodos generados** (no es necesario devolver el camino solución).
- Con la configuración propuesta, evaluar estrategias de **ANCHURA** y **PROFUNDIDAD** (pudiendo indicar nivel máximo de profundidad).
- Probar **diferentes evaluaciones** (nivel y otras configuraciones del escenario).

RECOMENDACIONES

- Probar inicialmente reglas sueltas.
- Probar ejecuciones paso-a-paso (ctrl-T)
- Probar Anchura/Profundidad con diversos niveles máximos.

EVALUACIÓN DE LA PRÁCTICA

- Evaluación por un examen **INDIVIDUAL** (última sesión práctica). ~~En cada grupo se realizarán dos turnos de examen, de 30 minutos aprox.~~

- El examen consistirá en preguntas sobre el código realizado, evaluación del SBR o realización de alguna modificación del código.

- Se generará una tarea en PoliformaT para la evaluación de la práctica.

- **ENTREGAS:** Cada alumno deberá subir a poliformat:
  - Código CLIPS realizado durante las sesiones de prácticas (versión antes del examen),
  - Código CLIPS modificado acorde a las instrucciones del examen. Respuestas.

etsinf

En resumen:

1. Diseñar e implementar el SBR ***(exploración en grafos)*** ***que resuelva el problema.***

   Diseño BH (búsqueda en grafo). Hechos iniciales con deffacts (estáticos, dinámicos)

   Diseño de reglas. Regla meta.

   No utilizar prioridades en las reglas salvo la prioridad de la regla de final.

   Representad el nivel en cada estado dinámico!

   Variable global: nodos generados.

   Función INICIO.

2. Realizar diferentes pruebas de evaluación para comparar los resultados con las estrategias de anchura y profundidad *(máxima profundidad)*.

   Contemplar distintas configuraciones.

   Obtener: nodos generados y nivel de la solución

ENTREGA: Programa CLISP con la implementación del SBR

EVALUACIÓN (individual): Evaluación del sistema, modificaciones del mismo, etc.