Implementación de GO en CLIPS

Sergio de Los Toyos, Iñigo Berasategi

https://github.com/sdlt2003/Go-board-game-in-CLIPS.git

0. Indice

1. Introducción y Descripción del Proyecto

El objetivo de este proyecto es desarrollar una implementación del juego de mesa 'GO' en CLIPS para el proyecto de la asignatura 'Inteligencia Artificial' de la EHU/UPV. CLIPS es un sistema experto, un lenguaje de programación que viene bien para este tipo de programas que tratan de simular el comportamiento humano por su estructura basada en hechos y reglas.

Para ello hemos desarrollado una interfaz compuesta por un tablero (que se nos venía prácticamente dado) y las distintas fichas con las que se va a poder jugar, además de las funciones y reglas necesarias para manipular el estado del juego conforme se avanza en la partida. Realizar este proyecto ha sido un desafío interesante, con varios problemas a resolver tanto a nivel programación como de organización. Estos problemas se explicarán en la última sección de este documento.

2. Configuración y Uso

Es importante que la versión de CLIPS utilizada a la hora de ejecutar el código sea CLIPS 6.31. Cualquier otra versión podría dar problemas tanto a la hora de inicializar el juego como en medio de la partida. De hecho, parte del código se había escrito originalmente para trabajar en CLIPS 6.41, lo que nos causó problemas para ejecutarlo en esta versión anterior.

Dicho esto, una vez descargada la versión correcta y descargado el archivo que viene junto a esta documentación (go.clp), habrá que cargar el archivo en el entorno de desarrollo CLIPS que el usuario tenga. Tras ejecutar el archivo con el comando `(run)`, el jugador tendrá que elegir tanto el tamaño del tablero (hay 3 tamaños disponibles: un 4×4, un 6×6 y un 9×9) como la naturaleza de los jugadores (si son humanos o IAs) y los colores de sus respectivas fichas. Una vez elegidos estos parámetros, dará comienzo el juego, donde el jugador tendrá que colocar la ficha poniendo la coordenada (x,y) de esta. Aquí un ejemplo gráfico de cómo debería de salir:

El jugador puede acabar la partida cuando lo desee. El programa está sujeto a las reglas del GO original, salvo que gana el que más fichas tenga en el tablero en vez de contar qué color ha ocupado el mayor espacio. El código está pensado para evitar errores de usuario como poner fichas fuera del tablero, movimientos ilegales, colocar fichas donde ya hay una colocada.

3. Documentación del Código

En esta parte nos centraremos en explicar la lógica tras el programa, no nos centraremos en explicar cada línea de código CLIPS. Al explicar cada una de las funciones, algunas se explicarán brevemente (sobre todo las más simples y las auxiliares), pero las que tienen más miga las analizaremos en más detalle.

Antes de comentar las funciones y reglas, repasemos rápidamente las variables globales y templates.

Variables globales

```
(defglobal ?*tamano* = 0)
```

Templates

Tablero

Plantilla para crear una variable multicampo ordenada de tablero. Varios de los parámetros de ésta no los hemos acabado usando al no llegar a programar modelos de máquina que expandan un arbol para tomar decisiones de jugadas.

```
(deftemplate tablero
    (slot id)
    (slot padre)
    (slot nivel)
    (multislot matriz)
)
```

Jugador

Plantilla para crear una variable multicampo ordenada del jugador.

```
(deftemplate jugador
    (slot id) ; 1/2, para saber quién va primero
    (slot tipo) ; humano/maquina
    (slot color) ; b/n
    (slot puntos) ; variable que acabamos sin usar
    (slot activo) ; variable booleana que usamos para el control de turnos
    (slot pass) ; variable de control que usamos para acabar el juego
)
```

Funciones

3.1 GenerarLineas1 y GenerarLineas2

- `generarLineas`: Imprime una línea separadora horizontal con un número de segmentos definidos por el argumento `?x`.
- `generarLineas2`: Imprime una línea con espacios entre segmentos para representar las líneas entre las filas del tablero.

```
(deffunction generarLineas (?x)
  (printout t crlf)
  (printout t "      |")
  (loop-for-count ?x
    (printout t "-----|")
  )
  (printout t crlf)
)

(deffunction generarLineas2 (?x)
  (printout t " ")
  (loop-for-count ?x
    (printout t "     |")
  )
  (printout t "     |")
  (printout t crlf)
)
```

3.2 Imprimir

`Imprimir`: Imprime el estado actual del tablero. Utiliza `generarLineas` y `generarLineas2` para estructurar visualmente el tablero y muestra las piezas blancas (B), negras (N) y espacios vacíos. Esta función y las anteriores comentadas son las que dan forma al tablero de juego.

```
(deffunction imprimir ($?mapeo)
    (printout t crlf)
    (printout t crlf)
    (loop-for-count (?i 0 ?*tamano*) do
      (if (= ?i 0) then
      (printout t "       ")
      else
      (printout t "  "?i "   "))
    )
    (generarLineas ?*tamano*)
    (loop-for-count (?fila 1 ?*tamano* ) do
      (generarLineas2 ?*tamano*)
      (printout t "   " ?fila "  |" )
      (loop-for-count (?columna 1 ?*tamano*) do
            (bind ?contenido (nth$  (+ (* ?*tamano* (- ?fila 1)) ?columna) $?mapeo))
            (if (eq ?contenido b) then
                (printout t  "  B  |")
            )
            (if (eq ?contenido n) then
                (printout t  "  N  |")
            )
            (if (eq ?contenido 0) then
                (printout t "     |")
            )
      )
      (generarLineas ?*tamano*)
```

```
    )
)
```

3.3 Fuera-De-Tablero, Pos-A-Cord y Cord-A-Pos

- `Fuera-de-tablero`: Verifica si una posición dada está fuera de los límites del tablero. Devuelve TRUE si la posición está fuera, y FALSE en caso contrario.
- `Pos-a-coord`: Convierte una posición en el tablero a coordenadas (x, y). Es útil para manejar las posiciones de manera más eficiente a lo largo del programa.
- `Coord-a-pos`: Convierte coordenadas (x, y) a una posición en el tablero, teniendo la función inversa a `Pos-a-coord`.

```
(deffunction fuera-de-tablero (?x ?y)
    (if (or (< ?x 1) (< ?y 1) (> ?x ?*tamano*) (> ?y ?*tamano*))
        then
        (return TRUE)
        else
        (return FALSE)
    )
)

(deffunction pos-a-coord (?pos)
    (bind ?x (mod (- ?pos 1) ?*tamano*)) ; Ajustar a índice basado en 1
    (bind ?y (+ 1 (div (- ?pos 1) ?*tamano*)))
    (return (create$ (+ ?x 1) ?y)) ; Ajustar la coordenada x para ser 1-basada
)


(deffunction coord-a-pos (?x ?y)
    (return (+ (* (- ?y 1) ?*tamano*) ?x))
)
```

3.4 Adyacente-A-Oponente y Ejes

- `adyacente-a-oponente`: Verifica si una posición en el tablero está adyacente a una pieza del color contrario al especificado. Sirve para determinar si una pieza es adyacente a la ficha del oponente.
- `ejes`: Calcula las posiciones adyacentes (norte, sur, este, oeste) a una posición en el tablero. Si una posición adyacente está fuera del tablero, devuelve f.

```
(deffunction adyacente-a-oponente (?pos ?ultimoColor $?mapeo)
    (bind ?contenido (nth$ ?pos $?mapeo))
    (return (neq ?contenido ?ultimoColor))
)

(deffunction ejes (?pos)
    (bind ?coords (pos-a-coord ?pos))
    (bind ?x (nth$ 1 ?coords))
    (bind ?y (nth$ 2 ?coords))
    (bind ?norte (if (not (fuera-de-tablero ?x (- ?y 1)))
                    then (coord-a-pos ?x (- ?y 1))
                    else f))
    (bind ?sur (if (not (fuera-de-tablero ?x (+ ?y 1)))
                    then (coord-a-pos ?x (+ ?y 1))
                    else f))
```

```
    (bind ?este (if (not (fuera-de-tablero (+ ?x 1) ?y))
                    then (coord-a-pos (+ ?x 1) ?y)
                    else f))
    (bind ?oeste (if (not (fuera-de-tablero (- ?x 1) ?y))
                    then (coord-a-pos (- ?x 1) ?y)
                    else f))
    (return (create$ ?norte ?sur ?este ?oeste))
```

3.5 Encrucijada y suelta

- `encruzijada`: Verifica si una posición en el tablero está completamente rodeada por piezas del oponente y no tiene espacios vacíos adyacentes.
- `suelta`: Determina si una posición en el tablero está 'suelta', es decir, si todas las posiciones adyacentes están vacías.

```
(deffunction encrucijada (?pos ?c1 $?mapeo)
    (bind ?ejes (ejes ?pos))
    (printout t "Ejes (dentro de encrucijada): " ?ejes crlf)
    (bind ?enc TRUE)
    (foreach ?eje ?ejes
        (if (neq ?eje f)
        then
            progn
            (if (or (eq (nth$ ?eje ?mapeo) 0) (eq (nth$ ?eje ?mapeo) ?c1))
            then
                (bind ?enc FALSE)
            )
        )
    )
    (return ?enc)
)


(deffunction suelta (?pos ?c1 $?mapeo)
    (bind ?ejes (ejes ?pos))
    (bind ?suelta TRUE)
    (foreach ?eje ?ejes
        (if (neq ?eje 0)
            then
            (bind ?suelta FALSE)
        )
    )
    return ?suelta
)
```

3.6 Rodea

`Rodea`: Determina si un grupo de piezas del oponente está completamente rodeado por las piezas del jugador y por tanto debe ser eliminado. Devuelve la lista de posiciones de las piezas a eliminar o FALSE si no hay piezas rodeadas.

```
; grupo := grupo de fichas adyacentes a la posición ?pos que se pueden comer
; (se trata como una variable multicampo porque )
; c     := color de las fichas que comen
(deffunction rodea (?grupo ?cEnemigas $?mapeo)
```

```
    (if (eq ?cEnemigas b) then
        (bind ?cAliadas n)
    else
        (bind ?cAliadas b)
    )

    (bind $?acomer (create$))

    (foreach ?pos ?grupo

        (bind ?enc (encrucijada ?pos ?cAliadas $?mapeo))
        (printout t "Resultado de ENCRUCIJADA en RODEA: " ?enc crlf)

        ; caso básico 1: ficha unica rodeada
        (if ?enc then
            (bind $?acomer (create$ $?acomer ?pos))
        else
            (printout t "Comprobando el caso general dentro de RODEA..." crlf)
            (bind ?visitados (create$ ?pos))
            (bind ?cola (create$ ?pos))
            (bind ?escape FALSE)

            (while (and (> (length$ ?cola) 0) (eq ?escape FALSE)) do
                (bind ?actual (nth$ 1 ?cola))
                (bind ?ejes (ejes ?actual))

                (printout t "Ejes de la ficha actual: " ?ejes crlf)

                (bind ?visitados (create$ ?visitados ?actual))
                (bind ?cola (delete-member$ ?cola ?actual))
                (bind ?i (nth$ 1 ?ejes))

                (while (> (length$ ?ejes) 0) do
                    (if (eq ?i f) then
                            (printout t "Eje " ?i " es f" crlf)
                    else
                        (if (eq (nth$ ?i ?mapeo) 0)
                            then
                            (bind ?escape TRUE)
                        else
                            (if (and (not (member$ ?i ?visitados)) (eq (nth$ ?i ?mape
o) ?cAliadas))
                                then
                                (bind ?visitados (create$ ?visitados ?i))
                                (bind ?cola (create$ ?cola ?i))
                            )
                        )
                    )
                    (bind ?ejes (delete-member$ ?ejes ?i))
                    (bind ?i (nth$ 1 ?ejes))
                )
                (bind ?cola (delete-member$ ?cola ?actual))
            )
            (if (eq ?escape FALSE)
                then
```

```
                (bind $?acomer (create$ $?acomer ?visitados))
            else
                (printout t "No se puede comer nada" crlf)
                (return FALSE)
            )
        )
    )
    (printout t "Resultado de RODEA (fichas que se van a comer): " $?acomer crlf)
    (printout t "" crlf)
    (return $?acomer)
)
```

3.7 Obtener-Adyacentes

`obtener-adyacentes`: Identifica todas las posiciones adyacentes a una posición dada que contienen piezas de un color específico.

```
; pos    := posición de la ultima ficha colocada
; c      := color de las fichas adyacentes que estás buscando
; $?mapeo:= tablero (basicamente)
(deffunction obtener-adyacentes (?pos ?c $?mapeo)
    (bind ?adyacentes (create$))   ; inicializamos
    ;(printout t "Posición de la ficha colocada (antes de pasar por pos-a-coord): " ?p
os crlf)
    (bind $?pos (pos-a-coord ?pos))
    ;(printout t "Posición de la ficha colocada (después de pasar por pos-a-coord): "
$?pos crlf)
    (bind ?x (nth$ 1 $?pos))       ; obtenemos la coordenada x
    (bind ?y (nth$ 2 $?pos))       ; obtenemos la coordenada y
    (loop-for-count (?dy -1 1) do               ; creamos coordenadas locales para pod
er recorrer los adyacentes
        (loop-for-count (?dx -1 1) do           ; de forma sencilla
            (if (or (neq ?dx 0) (neq ?dy 0))    ; excluimos la posición actual
                then
                (progn
                    (bind ?nx (+ ?x ?dx))       ; calculamos la posicion real del adya
cente que estamos calculando
                    (bind ?ny (+ ?y ?dy))
                    (if (not (fuera-de-tablero ?nx ?ny)) then   ; si no está fuera del
tablero
                        ;(printout t "Coordenadas del adyacente con coordenadas locale
s " ?dx " " ?dy ": " ?nx " " ?ny crlf)
                        (bind ?nPos (coord-a-pos ?nx ?ny))
                        ;(printout t "Posición real del adyacente: " ?nPos crlf)
                        (bind ?est (nth$ ?nPos $?mapeo))
                        (if (eq ?est ?c) then              ; si es del color del jugad
or
                            (bind ?adyacentes (create$ ?adyacentes ?nPos))  ; lo añadi
mos a la lista de adyacentes
                        )
                    else
                        ;(printout t "Adyacente analizando fuera de tablero" crlf)
                    )
                )
            )
```

```
        )
    )
    (return ?adyacentes)
)
```

3.8 Grupo

Esta función sirve de conector entre **comer** y **rodea,** devolviendo una lista de fichas que se deven comer, en caso de que se forme un grupo de fichas a comer. Si no, devuelve false.

```
(deffunction grupo (?pos ?color1 $?mapeo)
    ; Inicializa el color del jugador y el oponente
    (if (eq ?color1 b) then
        (bind ?color2 n)
    else
        (bind ?color2 b)
    )

    (printout t "Obteniendo fichas enemigas adyacentes a la ficha colocada..." crlf)
    (bind ?grupoEnemigo (obtener-adyacentes ?pos ?color2 $?mapeo))
    (printout t "Adyacentes: " ?grupoEnemigo crlf)

    (if (neq (length$ ?grupoEnemigo) 0)
        then
        (progn
            ; Verifica si las fichas capturables están rodeadas por el grupo
            (printout t "Entrando en RODEA para verificar si se come alguna ficha..."
crlf)
            (printout t "" crlf)

            (bind ?res (rodea ?grupoEnemigo ?color1 $?mapeo))
            (if (neq ?res FALSE)
                then
                    (printout t "Se debe(n) comer ficha(s)" crlf)
                    (return ?res)
                else
                    (printout t "No se debe comer nada" crlf)
                    (return FALSE)
            )
        )
    )
    (return FALSE)
)
```

3.9 Comer

`comer`: Verifica si un grupo de piezas enemigas adyacentes a una posición debe ser eliminado del tablero y actualiza el estado del tablero en consecuencia.

```
(deffunction comer (?pos ?c $?mapeo)
    (printout t "Entrando en grupo para ver si tocamos alguna ficha adyacente enemiga"
crlf)
    (printout t "" crlf)
```

```
    (bind ?res (grupo ?pos ?c $?mapeo)) ; tengo que conseguir que res sea una lista de
posiciones de fichas a eliminar
    (if (neq ?res FALSE) ; IF no es FALSE, entonces
        then
        (progn
            (printout t "Se van a eliminar las fichas de las siguientes posiciones: "
?res crlf)
            (foreach ?p ?res
                (bind $?mapeo (replace$ $?mapeo ?p ?p 0))
            )
            (return $?mapeo)
        )
        else
        ;(printout t "No se elimina ninguna ficha" crlf)
        (return FALSE)
    )
)
```

3.10 Ganador

`ganador`: Cuenta las piezas blancas y negras en el tablero para determinar el ganador del juego.

```
(deffunction ganador (?mapeo)
    (bind ?puntosB 0)
    (bind ?puntosN 0)
    (loop-for-count (?i 1 (* ?*tamano* ?*tamano*)) do
        (bind ?contenido (nth$ ?i ?mapeo))
        (if (eq ?contenido b) then
            (bind ?puntosB (+ ?puntosB 1))
        )
        (if (eq ?contenido n) then
            (bind ?puntosN (+ ?puntosN 1))
        )
    )
    (if (> ?puntosB ?puntosN) then
        (return "las blancas")
    )
    (if (> ?puntosN ?puntosB) then
        (return "las negras")
    )
    (return "oh! empate")
)
```

Reglas

3.10 Inicio

`inicio`: Configura el estado inicial de la partida. Solicita al usuario el tamaño del tablero y los tipos y colores de los jugadores, inicializando el tablero y los jugadores en consecuencia.

```
(defrule inicio
    (not (tablero))
=>
```

```
    (printout t "Inserte el tamaño (A) del tablero (AxA). Escoga entre 4, 6 u 9): " cr
lf)
    (bind ?aux_t (read))

    ; inicializacion de tablero
    (bind ?*tamano* ?aux_t)
    (bind ?*tamano* ?aux_t)
    (if (eq ?aux_t 4) then
        (assert (tablero (id 0) (padre -1) (nivel 0) (matriz 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0)))
    )
    (if (eq ?aux_t 6) then
        (assert (tablero (id 0) (padre -1) (nivel 0) (matriz 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)))
    )
    (if (eq ?aux_t 9) then
        (assert (tablero (id 0) (padre -1) (nivel 0) (matriz 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)))
    )

    (printout t "Inserte tipo del jugador 1 (h/m) : " crlf)
    (bind ?tipo (read))
    (printout t "Inserte color del jugador 1 (b/n) : " crlf)
    (bind ?color (read))

    ; inicializacion de jugador1
    (if (eq ?tipo h) then
        (if (eq ?color b) then
            (assert (jugador (id 1) (tipo h) (color b) (puntos 0) (activo TRUE)))
        )
        (if (eq ?color n) then
            (assert (jugador (id 1) (tipo h) (color n) (puntos 0) (activo TRUE)))
        )
    )
    (if (eq ?tipo m) then
        (if (eq ?color b) then
            (assert (jugador (id 1) (tipo m) (color b) (puntos 0) (activo TRUE)))
        )
        (if (eq ?color n) then
            (assert (jugador (id 1) (tipo m) (color n) (puntos 0) (activo TRUE)))
        )
    )

    (printout t "Inserte tipo del jugador 2 (h/m) : " crlf)
    (bind ?tipo (read))
    (printout t "Inserte color del jugador 2 (b/n) : " crlf)
    (bind ?color (read))

    ; inicializacion de jugador2
    (if (eq ?tipo h) then
        (if (eq ?color b) then
            (assert (jugador (id 2) (tipo h) (color b) (puntos 0) (activo FALSE) (pass
FALSE)))
        )
```

```
        (if (eq ?color n) then
            (assert (jugador (id 2) (tipo h) (color n) (puntos 0) (activo FALSE) (pass
FALSE)))
        )
    )
    (if (eq ?tipo m) then
        (if (eq ?color b) then
            (assert (jugador (id 2) (tipo m) (color b) (puntos 0) (activo FALSE) (pass
FALSE)))
        )
        (if (eq ?color n) then
            (assert (jugador (id 2) (tipo m) (color n) (puntos 0) (activo FALSE) (pass
FALSE)))
        )
    )
)
```

3.11 Mov

`mov`: Gestiona los movimientos de los jugadores humanos.

```
(defrule mov
    ?j <- (jugador (id ?i) (tipo h) (color ?c) (puntos ?puntos) (activo TRUE) (pass ?p
ass))
    ?tab <- (tablero (matriz $?mapeo))
=>
    (bind ?movimientoValido FALSE)
    (while (eq ?movimientoValido FALSE) do
        (printout t (if (eq ?i 1) then "Jugador 1, " else "Jugador 2, ") "ingresa tu m
ovimiento (x y) o acaba la partida (p p):")
        (bind ?x (read))
        (bind ?y (read))

        (if (and (eq ?x p) (eq ?y p))
            then
                (printout t "Acabando la partida..." crlf)
                (modify ?j (pass TRUE))
                (return)
        )

        (bind ?pos (+ (* ?*tamano* (- ?y 1)) ?x))
        (printout t "Posicion jugada: " ?pos crlf)

        (printout t "Verificando si el movimiento es válido..." crlf)
        (bind ?enc (encrucijada ?pos ?c $?mapeo))
        (printout t "Resultado de ENCRUCIJADA: " ?enc crlf)
        (bind ?est (nth$ ?pos $?mapeo))

        (if (and (and (eq ?enc FALSE) (eq ?est 0)) (and (<= ?x ?*tamano*) (<= ?y ?*tam
ano*)))
            then
                (printout t "Movimiento valido." crlf)
                (bind $?mapeo (replace$ $?mapeo ?pos ?pos (if (eq ?c b) then b else
n)))
                (retract ?tab)
```

```
                (assert (tablero (matriz $?mapeo)))

                ; Llamar a comer para verificar y eliminar fichas rodeadas
                (printout t "Entrando en comer para verificar y eliminar fichas rodead
as" crlf)
                (bind ?nuevoMapa (comer ?pos ?c $?mapeo))

                (if (neq ?nuevoMapa FALSE)
                    then
                        (retract ?tab)
                        (assert (tablero (matriz ?nuevoMapa)))
                        (printout t "Fichas comidas, actualizando tablero." crlf)
                )
                (bind ?movimientoValido TRUE)
            else
                (printout t "Movimiento invalido. Intente de nuevo." crlf)
                (printout t "----------------------------------------" crlf)
                (printout t " " crlf)
        )
    )

    (printout t "Tablero después del movimiento del jugador actual: " crlf)
    (if (neq ?nuevoMapa FALSE) then
        (imprimir ?nuevoMapa)
    else
        (imprimir $?mapeo)
    )
    (printout t "----------------------------------------" crlf)

    ;; Cambiar la activación del jugador
    (do-for-fact ((?juga jugador)) (eq ?juga:activo FALSE)
        (modify ?juga (activo TRUE))
    )
    (modify ?j (activo FALSE))

    (printout t "Turno completado." crlf)
)
```

3.12 Mov-Máquina

`mov-maquina`: Gestiona los movimientos de la IA. La máquina selecciona aleatoriamente una posición en el tablero donde podría colocar su pieza y verifica si la posición seleccionada es válida.

```
(defrule mov-maquina
    ?j <- (jugador (id ?i) (tipo m) (color ?c) (puntos ?puntos) (activo TRUE) (pass ?p
ass))
    ?tab <- (tablero (matriz $?mapeo))
=>
    (printout t "Turno de la máquina..." crlf)
    (bind ?aux FALSE)
    (bind ?lol 0)
    (while (eq ?aux FALSE)
        (bind ?mov (random 1 (* ?*tamano* ?*tamano*)))
        (printout t "Posición seleccionada por la máquina: " ?mov crlf)
        (if (and (eq (nth$ ?mov $?mapeo) 0) (eq (encrucijada ?mov ?c $?mapeo) FALSE))
```

```
            then
            (bind ?aux TRUE)
        )
        (bind ?lol (+ ?lol 1))
        (if (eq ?lol 200)
            then
            (printout t "La máquina no puede realizar un movimiento válido. Pasando el
turno..." crlf)
            (modify ?j (pass TRUE))
            break
        )
    )

    (if (eq ?aux TRUE) then
        (printout t "Posición jugada por la máquina: " ?mov crlf)

        (bind $?mapeo (replace$ $?mapeo ?mov ?mov (if (eq ?c b) then b else n)))
        (retract ?tab)
        (assert (tablero (matriz $?mapeo)))

        ; Llamar a comer para verificar y eliminar fichas rodeadas
        ;(printout t "Entrando en comer para verificar y eliminar fichas rodeadas" crl
f)
        (bind ?nuevoMapa (comer ?mov ?c $?mapeo))
        ;(printout t "Saliendo de comer. Contenido devuelto: " ?nuevoMapa crlf)
        (if (neq ?nuevoMapa FALSE)
            then
            (progn
                (retract ?tab)
                (assert (tablero (matriz ?nuevoMapa)))
                (imprimir ?nuevoMapa)
            )
        else
            (printout t "Tablero después del movimiento de la máquina: " crlf)
            (imprimir $?mapeo)
        )

        ;; Cambiar la activación del jugador
        (do-for-fact ((?juga jugador)) (eq ?juga:activo FALSE)
            (bind ?ident ?juga)
        )
        (modify ?ident (activo TRUE))
        (modify ?j (activo FALSE))
    )
)
```

3.13 Fin

`fin`: Determina el final del juego cuando un jugador decide pasar su turno, imprime el ganador y termina el juego.

```
(defrule fin
    (declare (salience 10))
    ?tab <- (tablero (matriz $?mapeo))
    ?j <- (jugador (id ?i) (tipo ?t) (color ?c) (puntos ?puntos) (activo ?act) (pass T
RUE))
```

```
=>
    (printout t "" crlf)
    (printout t "----------------------------------------" crlf)
    (printout t "" crlf)

    (printout t "Juego terminado." crlf)

    (printout t "" crlf)
    (printout t "----------------------------------------" crlf)
    (printout t "" crlf)

    (bind ?p1 (ganador ?mapeo))
    (printout t "Ganador: " ?p1 "!" crlf)

    (halt)
)
```

4. Conclusiones

Este proyecto ha sido un desafío a varios niveles. En primer lugar, ha sido difícil encontrar tiempo suficiente para dedicar al proyecto, ya que ambos miembros del equipo teníamos múltiples compromisos, tanto académicos como personales. Dado que todo el código está contenido en un solo archivo y la comprensión de la lógica del código es fundamental para avanzar, no pudimos dividir las tareas de la manera habitual de "tú haces esto, yo hago aquello", ya que cada función juega un papel fundamental en el resto.

Esta circunstancia ha retrasado considerablemente el progreso del proyecto en comparación a otros proyectos que hemos realizado, donde a lo mejor uno se encargaba de la interfaz gráfica, otro de la base de datos. Además, el hecho de que CLIPS carezca de una comunidad sólida en línea que pueda resolver nuestras dudas ha sido un inconveniente significativo. No solo no estábamos familiarizados con esta herramienta, sino que también es un lenguaje notablemente diferente a otros como Python, Java o C.

El heurístico utilizado en la inteligencia artificial del proyecto es bastante rudimentario, ya que juega de manera pseudoaleatoria. Nos hubiese gustado implementar una IA más sofisticada, pero no ha sido posible en esta ocasión. En general, nos hubiese gustado que la práctica se hubiese centrado en lenguajes de programación más modernos como Python, pero ha estado curiosa la experiencia.