**UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS**

(Universidad del Perú, Decana de América)

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**



**Inteligencia Artificial**

**Tema:** Implementación del juego "Los Gatos y el ratón” como un juego inteligente.

**Profesor:** Rolando Maguiña Pérez

**Integrantes:**

Palpán Flores, Yudely Guadalupe 15200138

Chávez Aliaga, Luis Ricardo 15200163

*Ciudad Universitaria, octubre del 2017*

Contenido

[Introducción 3](#_Toc495996810)

[Resumen 4](#_Toc495996811)

[Los gatos y el ratón 4](#_Toc495996812)

[Definición Formal del problema como un espacio de estados 5](#_Toc495996813)

[Problema 5](#_Toc495996814)

[Objetos 5](#_Toc495996815)

[Estado 5](#_Toc495996816)

[Estado inicial 6](#_Toc495996817)

[Estado final 7](#_Toc495996818)

[Reglas 7](#_Toc495996819)

[Estrategia MINIMAX 9](#_Toc495996820)

[Función de Evaluación 9](#_Toc495996821)

[Algoritmo minimax 9](#_Toc495996822)

[Aplicación 10](#_Toc495996823)

# Introducción

Los videojuegos han causado un gran impacto en la cultura popular desde su aparición en la década de los 50, incrementando cada vez más su calidad gráfica, los recursos para ejecutarse e incluso la historia que te pueden llegar a contar.

El avance de los videojuegos requirió que los personajes con los que interactúa el usuario tengan un cierto grado de inteligencia, para responder ante las distintas jugadas que pueda realizar jugador y darle agregarle un cierto grado de dificultad al competir con la máquina.

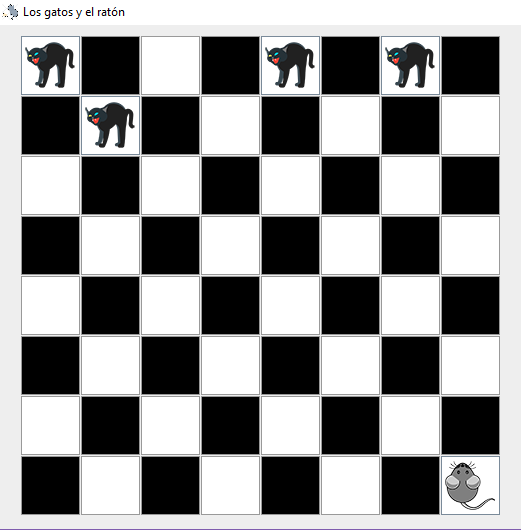
En el presente trabajo investigaremos la forma como podemos implementar el juego “Los gatos y el ratón”, definiéndolo como una búsqueda en un espacio de estados y su implementación en el lenguaje de alto nivel denominado Common Lisp.

# Resumen

En el presente informe se presenta el juego llamado “Los gatos y el Ratón” mediante un método informado de búsqueda, llamado estrategia MINIMAX. En primer lugar se presenta el problema relacionado al juego, seguidamente presentaremos su definición formal como un espacio de estados; enseguida el código fuente documentado que implementa dicho método. Por último, en la parte final se presentarán una serie de conclusiones y recomendaciones de cómo se podría mejorar el problema e implementación.

# Los gatos y el ratón

Los gatos y el ratón es un juego de información perfecta, se juega por turnos con dos participantes. Consiste en que un jugador mueve los gatos y el otro mueve al ratón; estos movimientos se realizan sobre un tablero de casillas blancas y negras. El objetivo de los gatos es atrapar al ratón de modo que este no tenga opción de moverse a ninguna casilla; el del ratón es “escaparse” de los gatos y lo logrará llegando a una línea más arriba de la línea del último gato.



# Definición Formal del problema como un espacio de estados

## Problema

Jugar “Los gatos y el ratón”

## Objetos

* Las fichas de los gatos
* La ficha del ratón
* El tablero

## Estado

Por lo que podemos representar el estado de la siguiente forma:

**(TAB, (x, y), T)**

Donde:

TAB: tablero

(x, y): posición de la ficha a mover

T: turno del jugador T є {1: gato, 9: ratón}

## Estado inicial

Configuración de las piezas en el tablero o dicho de otra manera la posición de cada ficha en el tablero.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **G** |  | **G** |  | **G** |  | **G** |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  | **R** |  |  |  |  |

Tablero actual\*

Representación Matricial\*

\* El ratón puede colocarse en cualquiera de los cuadrados blanco de su primera fila 9 (La posición de 9 puede ser i = 7 j є {1, 3, 5 7} dentro de la matriz que representa al tablero)

La lista de estados iniciales sería:

1. (TAB, (0, 0), 1)
2. (TAB, (0, 2), 1)
3. (TAB, (0, 4), 1)
4. (TAB, (0, 6), 1)
5. (TAB, (7, 1), 9)
6. (TAB, (7, 3), 9)
7. (TAB, (7, 5), 9)
8. (TAB, (7, 7), 9)

## Estado final

Hay dos posibles estados finales:

1. (TAB, (i, j), 9) donde:
2. (TAB, (i, j), 1) donde:

(ir, jr) = Posición del ratón obtenida de la matriz TAB.

* 1. ir - 1є [0, 7] y jr - 1є [0, 7] y TAB [ir, jr] = 1
  2. ir + 1є [0, 7] y jr - 1є [0, 7] y TAB [ir, jr] = 1
  3. ir + 1є [0, 7] y jr + 1є [0, 7] y TAB [ir, jr] = 1
  4. ir - 1є [0, 7] y jr - 1є [0, 7] y TAB [ir, jr] = 1

## Reglas

La notación que usaremos para el detalle de las reglas será:

* TAB: Matriz que representa tablero, inicializada en la configuración inicial.
* (i, j): Dupla que representa la posición de la jugada
* T: Jugador que está ejecutando la acción

|  |  |  |
| --- | --- | --- |
| Regla | Restricción | Nuevo Estado |
| AvzDerechaGato (TAB, (i, j), T) | (i < 7 or j < 7) and T =1 and TAB [i + 1, j + 1]=0 | (TAB, (i + 1, j + 1), 9)  TAB [i,j] = 0  TAB [i+1, j + 1] = 1 |
| AvzIzquerdaGato  (TAB, (i, j), T) | (i < 7 or j > 0 ) and T =1 and TAB [i + 1, j – 1]=0 | (TAB, (i + 1, j - 1), 9)  TAB [i,j] = 0  TAB [i+1, j – 1] = 1 |
| AvzDerechaRaton  (TAB, (i, j), T) | (i > 0 or j < 7) and T =9 and TAB [i - 1, j + 1]=0 | (TAB, (i - 1, j + 1), 1)  TAB [i,j] = 0  TAB[i-1, j + 1] = 9 |
| AvzIzquerdaRaton (TAB, (i, j), T) | (i > 0 or j > 0) and T = 9 and TAB [i - 1, j – 1]=0 | (TAB, (i - 1, j - 1), 1)  TAB [i,j] = 0  TAB[i-1, j – 1] = 9 |
| RtrDerechaRaton  (TAB, (i, j), T) | (i < 7 or j > 0 ) and T = 9 and TAB [i + 1, j – 1]=0 | (TAB, (i + 1, j - 1), 1)  TAB [i,j] = 0  TAB [i+1, j - 1) = 9 |
| RrtIzquerdaRaton  (TAB, (i, j), T) | (i < 7 or j < 7) and T = 9 and TAB [i + 1 , j + 1]=0 | (TAB, (i + 1, j + 1), 1)  TAB [i, j] = 0  TAB [i+1, j + 1] = 9 |

1. **AvzDerechaGato (TAB, (i, j), T):** Permite que el gato realice el movimiento de avanzar una posición a la derecha de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea menor a 7 o que j sea menor a 7, además de que T sea igual a 1 y TAB [i + 1, j + 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i + 1, j +1] sea igual a 1 y T sea igual a 9.
2. **AvzIzquierdaGato (TAB, (i, j), T):** Permite que el gato realice el movimiento de avanzar una posición a la izquierda de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea menor a 7 o que j sea mayor a 0, además de que T sea igual a 1 y TAB [i + 1, j - 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i + 1, j - 1] sea igual a 1 y T sea igual a 9.
3. **AvzDerechaRaton (TAB, (i, j), T):** Permite que el ratón realice el movimiento de avanzar una posición a la derecha de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea mayor a 7 o que j sea menor a 7, además de que T sea igual a 9 y TAB [i -1 1, j + 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i - 1, j + 1] sea igual a 9 y T sea igual a 1.
4. **AvzIquierdaRaton (TAB, (i, j), T):** Permite que el ratón realice el movimiento de avanzar una posición a la izquierda de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea mayor a 0 o que j sea mayor a 0, además de que T sea igual a 9 y TAB [i -1 1, j - 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i - 1, j - 1] sea igual a 1 y T sea igual a 1.
5. **RtrDerechaRaton (TAB, (i, j), T):** Permite que el ratón realice el movimiento de retroceder una posición a la derecha de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea menor a 7 o que j sea mayor a 0, además de que T sea igual a 9 y TAB [i + 1, j - 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i + 1, j - 1] sea igual a 9 y T sea igual a 1.
6. **RtrIzquierdaRaton (TAB, (i, j), T):** Permite que el ratón realice el movimiento de retroceder una posición a la izquierda de su posición actual (representado por (i, j)). Las condiciones para que se realice son de que el índice de TAB i sea menor a 7 o que j sea menor a 7, además de que T sea igual a 9 y TAB [i + 1, j + 1] sea igual a 0. El resultado es que TAB [i, j] sea igual 0, TAB [i + 1, j +1] sea igual a 9 y T sea igual a 1.

# Estrategia MINIMAX

## Función de Evaluación

## Algoritmo minimax

# Aplicación

A continuación presentamos el código fuente documentado del juego “Los gatos y el ratón” generado con el lenguaje Common Lisp.

## Código fuente en Common Lisp:

*;Establecemos el estado inicial del tablero en la variable “tablero”.*

(setq tablero

'(

(1 0 1 0 1 0 1 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)

)

)

*;Se establece el estado actual que estará conformado por una lista (tablero turno)*

(setq estado-actual (list tablero 1))

*;Función para establecer el estado actual*

(defun estado-actual ()

estado-actual

)

(defun to-pos (x y)

(list x y)

)

*,Función que establece aleatoriamente la posición del ratón en la primera fila del tablero*

(defun elegir-pos ()

(let ((pos (random 8)))

(if (oddp pos) pos (elegir-pos))

)

)

(defun iniciar-nuevo-juego (turno)

(let ((copia-tablero (copy-tree tablero)))

(setf (nth (elegir-pos) (nth 7 copia-tablero)) 9)

(setq estado-actual (list copia-tablero turno))

)

)

*;Función que obtiene el estado actual*

(defun set-eactual (estado)

(setq estado-actual estado)

)

; Turno actual

(setq tr 1)

*;Función que define las jugadas del gato*

*; i. Función que define la jugada avanzar a la diagonal derecha, siendo el turno del gato*

(defun avanzar-gato-derecha (estado p-ficha-mov)

(let

(

*; La funcion copy-tree crea una copia pero no liga las variables*

(tab (copy-tree (nth 0 estado)))

*; Se establece el turno del jugador en la variable TURNO*

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov)) *; Pos i de la ficha*

(j (nth 1 p-ficha-mov)) *; Pos j de la ficha*

)

(cond

(

(and (and (< i 8) (< j 8))

(and (< (+ i 1) 8) (< (+ j 1) 8))

(= turno 1)

(= (nth (+ j 1) (nth (+ i 1) tab)) 0)

(= (nth j (nth i tab)) 1)

)

(setf (nth (+ j 1) (nth (+ i 1) tab)) 1)

(setf (nth j (nth i tab)) 0)

(list tab 9)

)

)

)

)

*; ii. Función que define la jugada avanzar a la diagonal izquierda, siendo el turno del gato*

(defun avanzar-gato-izquierda (estado p-ficha-mov)

(let

(

(tab (copy-tree (nth 0 estado)))

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov))

(j (nth 1 p-ficha-mov))

)

(cond

(

(and (and (< i 8) (>= j 0))

(and (< (+ i 1) 8) (>= (- j 1) 0))

(= turno 1)

(= (nth (- j 1) (nth (+ i 1) tab)) 0)

(= (nth j (nth i tab)) 1)

)

(setf (nth (- j 1) (nth (+ i 1) tab)) 1)

(setf (nth j (nth i tab)) 0)

(list tab 9)

)

)

)

)

*; Función que define las jugadas del ratón*

*; i. Función que define la jugada avanzar a la diagonal derecha, siendo el turno del ratón.*

(defun avanzar-raton-derecha (estado p-ficha-mov)

(let

(

(tab (copy-tree (nth 0 estado)))

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov))

(j (nth 1 p-ficha-mov))

)

(cond

(

(and (and (>= i 0) (< j 8))

(and (>= (- i 1) 0) (< (+ j 1) 8))

(= turno 9)

(= (nth (+ j 1) (nth (- i 1) tab)) 0)

(= (nth j (nth i tab)) 9)

)

(setf (nth (+ j 1) (nth (- i 1) tab)) 9)

(setf (nth j (nth i tab)) 0)

(list tab 1)

)

)

)

)

*; ii. Función que define la jugada avanzar a la diagonal izquierda, siendo el turno del ratón.*

(defun avanzar-raton-izquierda (estado p-ficha-mov)

(let

(

(tab (copy-tree (nth 0 estado)))

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov))

(j (nth 1 p-ficha-mov))

)

(cond

(

(and (and (>= i 0) (>= j 0))

(and (>= (- i 1) 0) (>= (- j 1) 0))

(= turno 9)

(= (nth (- j 1) (nth (- i 1) tab)) 0)

(= (nth j (nth i tab)) 9)

)

(setf (nth (- j 1) (nth (- i 1) tab)) 9)

(setf (nth j (nth i tab)) 0)

(list tab 1)

)

)

)

)

*; iii. Función que define la jugada retroceder a la diagonal derecha, siendo el turno del ratón*

(defun retro-raton-derecha (estado p-ficha-mov)

(let

(

(tab (copy-tree (nth 0 estado)))

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov))

(j (nth 1 p-ficha-mov))

)

(cond

; Primera condicion

(

; Comprobacion

(and (and (< i 8) (< j 8))

(and (< (+ i 1) 8) (< (+ j 1) 8))

(= turno 9)

(= (nth (+ j 1) (nth (+ i 1) tab)) 0)

(= (nth j (nth i tab)) 9)

)

; Acciones

(setf (nth (+ j 1) (nth (+ i 1) tab)) 9)

(setf (nth j (nth i tab)) 0)

(list tab 1)

)

)

)

)

*; iv. Función que define la jugada retroceder a la diagonal izquierda, siendo el turno del ratón.*

(defun retro-raton-izquierda (estado p-ficha-mov)

(let

(

(tab (copy-tree (nth 0 estado)))

(turno (nth 1 estado))

(i (nth 0 p-ficha-mov))

(j (nth 1 p-ficha-mov))

)

(cond

(

; Comprobacion

(and (and (< i 8) (>= j 0))

(and (< (+ i 1) 8) (>= (- j 1) 0))

(= turno 9)

(= (nth (- j 1) (nth (+ i 1) tab)) 0)

(= (nth j (nth i tab)) 9)

)

; Acciones

(setf (nth (- j 1) (nth (+ i 1) tab)) 9)

(setf (nth j (nth i tab)) 0)

(list tab 1)

)

)

)

)

*; Función que crea la vista del tablero, como una representación matricial, para ser visualizada en consola.*

(defun vista-jugada (estado)

(cond

((not estado) "Jugada no valida")

((not (not estado))

(imprimir-tablero (nth 0 estado))

(format t "~&Turno: ~S" (nth 1 estado))

)

)

)

*; Función que define el formato del tablero como una matriz.*

(defun imprimir-tablero (tablero)

(dolist (fila tablero)

(format t "~&~S" fila)

)

)

*; Función que suma los valores de las filas*

(defun suma-fila (fila)

((lambda (sum)

(mapcar #' (lambda (i) (setq sum (+ sum i))) fila)

sum

) 0)

)

*; Función que suma filas delante del ratón*

(defun sumar-filas-adelante (tablero n)

((lambda (sum)

(dotimes (i n) (setq sum (+ sum (suma-fila (nth i tablero)))))

sum

) 0)

)

*; Función que encuentra la posición del ratón en el tablero y la asocia a la variable “pos-ratón”.*

(defun pos-raton (tablero)

((lambda (x y)

(dotimes (i 8)

(dotimes (j 8)

(cond

((= (nth j (nth i tablero)) 9)

(setq x i)

(setq y j)

)

)

)

)

(list x y)

) 0 0)

)

*; Función que encuentra la posición de cada gato en el tablero y los asocia a la lista “pos-gatos”.*

(defun pos-gatos (tablero)

((lambda (pos-gatos)

(dotimes (i 8)

(dotimes (j 8)

(cond

((= (nth j (nth i tablero)) 1)

(setq pos-gatos (append pos-gatos (list (list i j))))

)

)

)

)

pos-gatos

) nil)

)

*; Función que evalúa los movimientos del ratón, si este ya no tiene lugar de avance establece al estado como NIL, en consecuencia al ratón como perdedor.*

(defun raton-encerrado (estado)

((lambda (p-raton e-alterno)

(setf (nth 1 e-alterno) 9)

(and (equal (avanzar-raton-izquierda e-alterno p-raton) nil)

(equal (avanzar-raton-derecha e-alterno p-raton) nil)

(equal (retro-raton-izquierda e-alterno p-raton) nil)

(equal (retro-raton-derecha e-alterno p-raton) nil))

) (pos-raton (nth 0 estado)) (copy-tree estado))

)

*; Función de parada, establece los estados finales del juego.*

*; Si turno es 9 -> T: Ha ganado el ratón, NIL: Ha perdido el ratón.*

*; Si turno es 1 -> T: Ha ganado el gato, NIL: Ha perdido el gato.*

(defun test-parada (estado)

((lambda (tab turno)

(cond

((= turno 9) (if (= (sumar-filas-adelante tab (nth 0 (pos-raton tab))) 0) t nil))

((= turno 1) (if (raton-encerrado estado) t nil))

)

) (copy-tree (nth 0 estado)) (nth 1 estado))

)

*; Función que valida estado ganador para el ratón.*

(defun es-ganador (estado)

(if (raton-encerrado estado) t nil)

)

*; Función que valida estado perdedor para el ratón.*

(defun es-perdedor (estado)

((lambda (tab)

(if (= (sumar-filas-adelante tab (nth 0 (pos-raton tab))) 0) t nil)

) (copy-tree (nth 0 estado)))

)

*; Función que genera los sucesores para cada estado, en la posición actual.*

(defun gen-sucesores (estado p-ficha-mov)

((lambda (sucesores)

(cond

*; Si el turno es del ratón, realizará un apilamiento de sucesores en una sola lista llamada “sucesores”, esto lo realiza evaluando cada regla que define sus movimientos.*

((= (nth 1 estado) 9)

(if (avanzar-raton-izquierda estado p-ficha-mov)

(setq sucesores (append sucesores (list (avanzar-raton-izquierda estado p-ficha-mov)))))

(if (avanzar-raton-derecha estado p-ficha-mov)

(setq sucesores (append sucesores (list (avanzar-raton-derecha estado p-ficha-mov)))))

(if (retro-raton-izquierda estado p-ficha-mov)

(setq sucesores (append sucesores (list (retro-raton-izquierda estado p-ficha-mov)))))

(if (retro-raton-derecha estado p-ficha-mov)

(setq sucesores (append sucesores (list (retro-raton-derecha estado p-ficha-mov)))))

)

*; Si el turno es del gato, realizará un apilamiento de sucesores en una sola lista llamada “sucesores”, esto lo realiza evaluando cada regla que define sus movimientos.*

((= (nth 1 estado) 1)

(if (avanzar-gato-izquierda estado p-ficha-mov)

(setq sucesores (append sucesores (list (avanzar-gato-izquierda estado p-ficha-mov)))))

(if (avanzar-gato-derecha estado p-ficha-mov)

(setq sucesores (append sucesores (list (avanzar-gato-derecha estado p-ficha-mov)))))

)

)

sucesores

) nil)

)

*; Función que asocia los sucesores generados por turno.*

(defun sucesores-por-turno (estado)

(let ((lista-sucesores nil))

(cond

((= (nth 1 estado) 1)

(dolist (p-gato (pos-gatos (nth 0 estado)))

(setq lista-sucesores (append lista-sucesores (gen-sucesores estado p-gato)))

)

)

((= (nth 1 estado) 9)

(setq lista-sucesores (gen-sucesores estado (pos-raton (nth 0 estado))))

)

)

lista-sucesores

)

)

*; Función que muestra los sucesores de estado actual.*

(defun mostrar-sucesores (sucesores)

(mapcar #' (lambda (sucesor) (vista-jugada sucesor)) sucesores)

)

*; FUNCIÓN DE EVALUACIÓN*

*; función potencia de un número*

(defun potencia (x)

(\* x x)

)

*; Función que calcula la distancia entre dos puntos*

(defun distancia (p1 p2)

(let ((x0 (nth 0 p1)) (y0 (nth 1 p1)) (x1 (nth 0 p2)) (y1 (nth 1 p2)))

(sqrt (+ (potencia (- x1 x0)) (potencia (- y1 y0))))

)

)

*; función que establece un valor para una determinada posición en el tablero.*

*; posición-evaluación = 1 - > el punto se encuentra encima del punto de referencia*

*; posición-evaluación = -1 - > el punto se encuentra debajo del punto de referencia*

(defun posicion-evaluacion (p1 p2)

(if (> (- (nth 0 p2) (nth 0 p1)) 0)

1

-1

)

)

*; función que calcula la pendiente asociada a dos puntos de referencia.*

(defun pendiente (p1 p2)

(let ((x1 (nth 0 p1)) (y1 (nth 1 p1)) (x2 (nth 0 p2)) (y2 (nth 1 p2)))

(if (= 0 (- x1 x2)) 0 (/ (- y1 y2) (- x1 x2)))

)

)

*; función que evalúa si un punto se encuentra en la misma diagonal de otro punto específico dentro del tblero..*

(defun misma-diagonal (p1 p2)

(if (= (abs (pendiente p1 p2)) 1) 10 0.5)

)

*; función que evalúa si un punto se encuentra en la misma columna de otro punto específico dentro del tablero..*

(defun misma-linea (p1 p2)

(if(= (nth 1 p1) (nth 1 p2)) 5 0.5)

)

*; Función de Evaluación*

(defun evaluacion (tablero)

(let ((dist-gato-raton 0) (p-raton (pos-raton tablero)))

(mapcar #'

(lambda (p-gato)

(setq dist-gato-raton

(+ dist-gato-raton

(\* (posicion-evaluacion p-gato p-raton) (distancia p-gato p-raton) (misma-linea p-gato p-raton) (misma-diagonal p-gato p-raton))

)

)

)

(pos-gatos tablero)

)

dist-gato-raton

)

)

(defun evaluar-sucesores (sucesores)

(mapcar #'

(lambda (sucesor)

(evaluacion (nth 0 sucesor))

)

sucesores

)

)

*; Función que realiza un ordenamiento a la lista de sucesores.*

(defun ordenar-sucesores (sucesores)

(let ((ltemp nil))

(mapcar #'

(lambda (sucesor)

(setq ltemp (append ltemp (list (list sucesor (evaluacion (nth 0 sucesor))))))

)

sucesores

)

(sort ltemp #'> :key #'cadr)

)

)

*; Función del Algoritmo MINIMAX*

*; función que obtiene el valor máximo*

(defun maximo (vals)

(car (sort vals #'>))

)

*; función que obtiene el valor mínimo*

(defun minimo (vals)

(car (sort vals #'<))

)

*; función que establece si es nodo MIN, de acuerdo al nivel, serán los de nivel con valor impar.*

(defun esNodoMIN (nivel)

(evenp nivel)

)

*; Función que establece si es nodo MAX, de acuerdo al nivel, serán los de nivel con valor par.*

(defun esNodoMAX (nivel)

(oddp nivel)

)

*; Función MINIMAX1*

*; jugador máquina : gato*

*; jugador humano : ratón*

(defun minimax-gato (estado nivel)

(cond

*; Al estado ganador le asocia un valor MAX*

((es-ganador estado) 99)

*; Al estado perdedor le asocia un valor MIN*

((es-perdedor estado) -99)

*; Establece hasta que nivel de conocimiento respecto a los sucesores generados obtendrá*

*; Ejecuta la función de evaluación en las ramas, de acuerdo al nivel establecido.*

((= nivel 5) (evaluacion (nth 0 estado)))

*; Realiza la función recursiva “minimax” asociando un valor Min o Max a cada sucesor y apilándolo en la variable “vals”.*

((< nivel 5)

(let ((vals nil))

(dolist (sucesor (sucesores-por-turno estado) )

(setq vals (append vals (list (minimax-gato sucesor (+ nivel 1)))))

)

(if (esNodoMAX nivel) (maximo vals) (minimo vals))

; (if (esNodoMIN nivel) (return (minimo vals)))

)

)

)

)

*; Función que elegirá el estado sucesor que tenga asociado a él el valor máximo o mínimo de acuerdo al turno.*

(defun eleccion-minimax (estado)

(let ((sucesores (sucesores-por-turno estado)) (ltemp nil))

(mapcar #'

(lambda (sucesor)

(setq ltemp (append ltemp (list (list sucesor (minimax-gato sucesor 1)))))

)

sucesores

)

(caar (sort ltemp #'> :key #'cadr))

)

)

*; Función MINIMAX2*

*; jugador máquina : ratón*

*; jugador humano : gato*

(defun minimax-raton (estado nivel)

(cond

((es-ganador estado) -99)

((es-perdedor estado) 99)

((= nivel 5) (\* -1 (evaluacion (nth 0 estado))))

((< nivel 5)

(let ((vals nil))

(dolist (sucesor (sucesores-por-turno estado) )

(setq vals (append vals (list (minimax-raton sucesor (+ nivel 1)))))

)

(if (esNodoMAX nivel) (maximo vals) (minimo vals))

; (if (esNodoMIN nivel) (return (minimo vals)))

)

)

)

)

*; Función que elegirá el estado sucesor que tenga asociado a él el valor máximo o mínimo de acuerdo al turno.*

(defun eleccion-minimax-raton (estado)

(let ((sucesores (sucesores-por-turno estado)) (ltemp nil))

(mapcar #'

(lambda (sucesor)

(setq ltemp (append ltemp (list (list sucesor (minimax-raton sucesor 1)))))

)

sucesores

)

(caar (sort ltemp #'> :key #'cadr))

)

)

*;EJECUTAR JUEGO EN CONSOLA*

*; para el jugador maquina:*

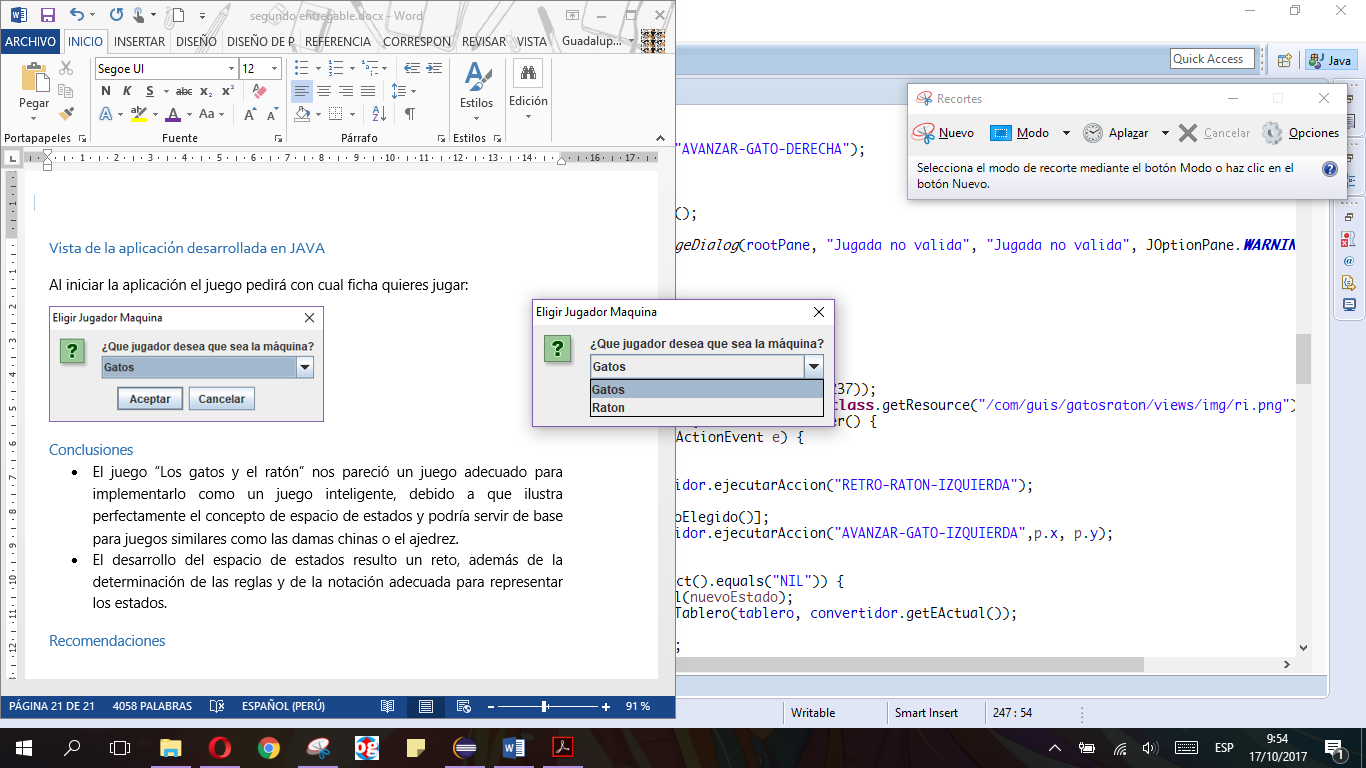
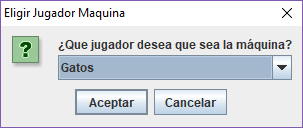
*; (vista-jugada (setq eactual (eleccion-minimax eactual)))*

*;para el jugador humando:*

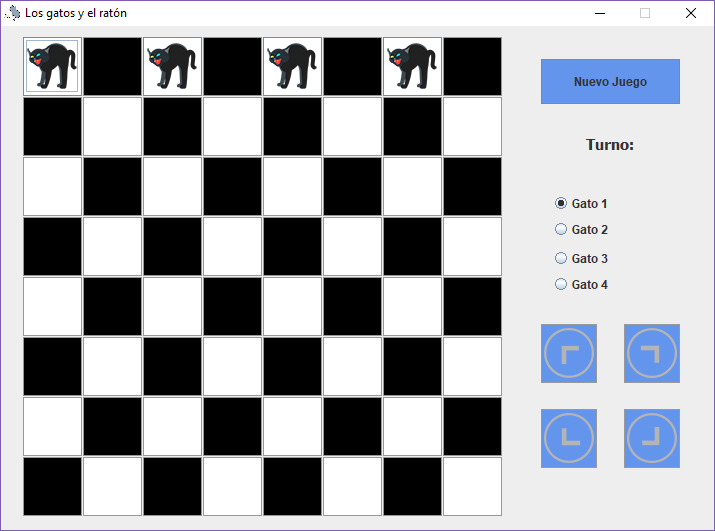
*; (vista-jugada (setq eactual (avanzar-raton-izquierda eactual (pos-raton (nth 0 eactual)))))*

## Vista de la aplicación desarrollada en JAVA

Al iniciar la aplicación el juego pedirá con cual ficha quieres jugar:



Una vez seleccionada la ficha, abrirá la interfaz principal, donde se muestra el botón iniciar juego.



# Conclusiones

* El juego “Los gatos y el ratón” nos pareció un juego adecuado para implementarlo como un juego inteligente, debido a que ilustra perfectamente el concepto de espacio de estados y podría servir de base para juegos similares como las damas chinas o el ajedrez.
* El desarrollo del espacio de estados resulto un reto, además de la determinación de las reglas y de la notación adecuada para representar los estados.

# Recomendaciones

# Bibliografía

Russell, S. J., & Norvig, P. (2004). *Inteligencia Artificial, un enfoque moderno.* Madrid: Person Prentice Hall.