Tic Tac Toe

Castillo Jara Edna Paola

Teoría Computacional. Universidad Politécnica de San Luis Potosí, México

Abstract-- En este documento se presenta el desarrollo del proyecto correspondiente a la materia de Teoría Computacional el cual consiste en el famoso y entretenido juego de Tic Tac Toe en donde se juega Máquina vs. Usuario, se implementa el código de programación en el lenguaje F# y se sigue una lógica que se explicará a lo largo de este documento. Así como algunos diagramas que permitan entender el funcionamiento del programa.

I. INTRODUCCIÓN

El famoso juego de Tic Tac Toe siempre ha sido un clásico y sencillo, pero entretenido, juego que siempre nos logra sacar del aburrimiento, probablemente se le conoce como el juego del "gato", o bien, tres en línea. Pero las reglas son las mismas. En esta ocasión se ha decidido hacer un poco más complejo el juego, con las mismas reglas y el mismo tablero, pero teniendo como contrincante a un ordenador. El reto será vencerlo y en este documento se hablará a fondo sobre la implementación y funcionamiento del código, así como de la lógica que se siguió para instruir correctamente al ordenador.

II. ANTECEDENTES

Se denomina inteligencia artificial a la rama de la ciencia informática dedicada al desarrollo de agentes racionales no vivos. es la disciplina que se encarga de construir procesos que al ser ejecutados sobre una arquitectura física producen acciones o resultados que maximizan una medida de rendimiento determinada, basándose en la secuencia de entradas percibidas y en el conocimiento almacenado en tal arquitectura. [1]

Debido a la gran demanda de tecnología, la IA es una ciencia aplicada que consiste en incluir inteligencia a máquinas creadas por el hombre, estas tecnologías de IA también son aplicadas en juegos para darle una mejor interacción humano-computador.

El algoritmo de minimax, por ejemplo. En simples palabras consiste en la elección del mejor movimiento para el computador, suponiendo que el contrincante escogerá uno que lo pueda perjudicar, para escoger la mejor opción este algoritmo realiza un árbol de búsqueda con todos los posibles movimientos, luego recorre todo el árbol de soluciones del juego a partir de un estado dado, es decir, según las casillas que ya han sido rellenadas. Por tanto, minimax se ejecutará cada vez que le toque mover a la IA.[2]

III. METODOLOGÍA

A. Sobre el juego

Para comenzar, es importante abordar nuevamente las reglas del juego.

- Se tienen dos jugadores, en este caso, computadora y humano.
- Se asigna un caracter a cada jugador (X o O).
- Se alterna un turno y un turno para cada jugador.
- Al juntar 3 Xs o 3 Os en diagonal, vertical u horizontal se declara ganador dicho jugador.
- Al llenarse las 9 casillas y no darse el caso anterior se declara un empate.

Así mismo, se tienen 8 combinaciones diferentes en las que se puede ganar, las cuales son:

i. 1, 2 y 3° posición.
ii. 4, 5 y 6° posición.
iii. 7, 8 y 9° posición.
iv. 1, 4 y 7° posición.
v. 2, 5 y 8° posición.
vi. 3, 6 y 9° posición.
vii. 1, 5 y 9° posición.
viii. 3, 5 y 7° posición.

Se puede apreciar en las siguientes figuras con mayor claridad.

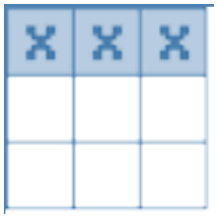

Figura 1. 1, 2 y 3° posición

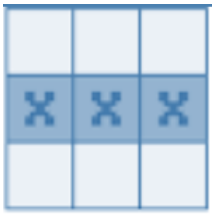

Figura 2. 4, 5 y 6° posición.

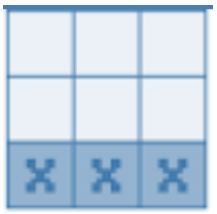

Figure 3. 7, 8 y 9° posición.

Figure 4. 1, 4 y 7° posición.

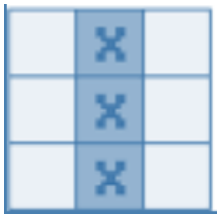

Figure 5. 2, 5 y 8° posición.

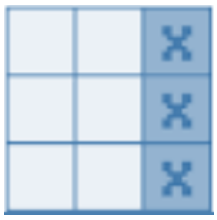

Figure 6. 3, 6 y 9° posición.

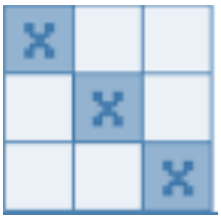

Figure 7. 1, 5 y 9° posición.

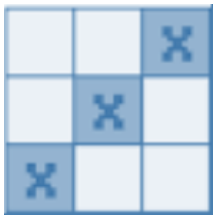

Figure 8. 3, 5 y 7° posición.

B. Algoritmo

- Condición terminal determina cuando el juego se acabó, ya sea cuando hay un ganador, un empate o un perdedor (1, 0, -1).
- Los intermedios del juego están dados por los turnos de cada jugador y los movimientos posibles y disponibles.
- Si es el turno de X (computadora) se implementa el máximo, es decir, busca movimientos que lo conduzcan al mayor número positivo.
- Si es el turno de O (usuario) se implementa el mínimo. En otras palabras, si un movimiento significa una pérdida para X, entonces la computadora lo evitaría.
- El jugador que espera valores positivos se conoce como maximizador.
- El jugador que espera valores negativos se conoce como minimizador.
- Los posibles valores de la función de utilidad tienen un rango de [1-9]

IV. DESARROLLO

A continuación, se mostrará la manera en la que se implementó el código. Para esto se utilizó como IDE Microsoft Visual Studio, en el lenguaje de programación F#, con la finalidad de que el programa pudiera ser compilado en cualquier sistema operativo, o en su mayoría. Para esto, es necesario haber instalado el software de Microsoft en el cual se codificó.

Se mostrarán las funciones principales que hacen funcionar el programa.

Para comenzar, se muestra la manera en la que se ha programado el tablero para tener un punto de partida e ir comprendiendo el desglose.

```
let render ((a, b, c), (d, e, f), (g, h, i)) =  // Every cell is a letter
  Console.ForegroundColor<-ConsoleColor.Blue  //cambia el color a azul
  sprintf
    """
  %s | %s | %s
  -----------
  %s | %s | %s
  -----------
  %s | %s | %s"""
    (renderValue a) (renderValue b) (renderValue c) //Prints the X or O in its cell
    (renderValue d) (renderValue e) (renderValue f)
    (renderValue g) (renderValue h) (renderValue i)
```

Figure 9. Función para imprimir el tablero

Se pueden observar las nueve celdas en donde se han de colocar las Xs y Os según sea el caso, tal como es en el juego tradicional.

Se muestran, ahora, las funciones emptyBoard, Position y Move. Además de algunas variables que se explican en seguida.

```
type Value =
  | Unspecified // To set it to the cells of the empty board
  | Letter of Letter  // To fill in the cells once they have been selected

type OneThroughThree = | One | Two | Three
type Row = Value*Value*Value
type Board = Row*Row*Row

let emptyBoard = //For the empty board
  (Unspecified, Unspecified, Unspecified), //This is the default behavior a
  (Unspecified, Unspecified, Unspecified),
  (Unspecified, Unspecified, Unspecified)

type Position = { //To convert the player input
  Column: OneThroughThree
  Row: OneThroughThree
}

type Move = {
  At: Position  // Player input that has to be converted
  Place: Letter // Correspons to WhoseTurn
}
```

Figure 10. Función emptyBoard.

En la función emptyBoard, como se puede intuir por su nombre, se asigna un valor que aún no ha sido especificado a las nueve casillas del tablero.

Además, con oneThroughThree se puede evaluar la selección de la casilla y determinar a qué posición en el tablero corresponde y se asigna a las variables row y column.

Así mismo, se muestran las funciones select y set que son de suma importancia.

```
let select (board: Board) (position: Position) =  //to verify if the posititon is empty
    match board, position with
    | ((x, _, _), _, _), { Row=One; Column=One } -> x
    | ((_, x, _), _, _), { Row=One; Column=Two } -> x
    | ((_, _, x), _, _), { Row=One; Column=Three } -> x
    | (_, (x, _, _), _), { Row=Two; Column=One } -> x
    | (_, (_, x, _), _), { Row=Two; Column=Two } -> x
    | (_, (_, _, x), _), { Row=Two; Column=Three } -> x
    | (_, _, (x, _, _)), { Row=Three; Column=One } -> x
    | (_, _, (_, x, _)), { Row=Three; Column=Two } -> x
    | (_, _, (_, _, x)), { Row=Three; Column=Three } -> x

let set value (board: Board) (position: Position) =
    match board, position with
    | ((_, v2, v3), r2, r3), { Row=One; Column=One } -> (value, v2, v3), r2, r3 // To assign the input once it has been verif
    | ((v1, _, v3), r2, r3), { Row=One; Column=Two } -> (v1, value, v3), r2, r3
    | ((v1, v2, _), r2, r3), { Row=One; Column=Three } -> (v1, v2, value), r2, r3
    | (r1, (_, v2, v3), r3), { Row=Two; Column=One } -> r1, (value, v2, v3), r3
    | (r1, (v1, _, v3), r3), { Row=Two; Column=Two } -> r1, (v1, value, v3), r3
    | (r1, (v1, v2, _), r3), { Row=Two; Column=Three } -> r1, (v1, v2, value), r3
    | (r1, r2, (_, v2, v3)), { Row=Three; Column=One } -> r1, r2, (value, v2, v3)
    | (r1, r2, (v1, _, v3)), { Row=Three; Column=Two } -> r1, r2, (v1, value, v3)
    | (r1, r2, (v1, v2, _)), { Row=Three; Column=Three } -> r1, r2, (v1, v2, value)
```

Figure 11. Funciones select y set.

La función select, al recibir el estado actual del tablero (disponibilidad de las casillas) y la posición seleccionada por el jugador en turno, verifica que dicha celda esté disponible.

Una vez que se ha aprobado dicho movimiento, continúa la función set para así asignar el símbolo (X o O) a la casilla seleccionada y evaluada anteriormente.

Ahora, se abordará la manera en la que se realiza dicha evaluación y cómo se recibe la casilla seleccionada por los jugadores (computadora o usuario).

Se tienen las siguientes dos funciones.

```
let waysToWin = // Possible combinations in which a player can win
  [
    { Row=One; Column=One }, { Row=One; Column=Two }, { Row=One; Column=Three } //  1, 2 and 3rd position.
    { Row=Two; Column=One }, { Row=Two; Column=Two }, { Row=Two; Column=Three }  //  4, 5 and 6th position.
    { Row=Three; Column=One }, { Row=Three; Column=Two }, { Row=Three; Column=Three } //  7, 8 and 9th position.

    { Row=One; Column=One }, { Row=Two; Column=One }, { Row=Three; Column=One } //  1, 4 and 7th position.
    { Row=One; Column=Two }, { Row=Two; Column=Two }, { Row=Three; Column=Two } //  2, 5 and 8th position.
    { Row=One; Column=Three }, { Row=Two; Column=Three }, { Row=Three; Column=Three }  //  3, 6 and 9th position

    { Row=One; Column=One }, { Row=Two; Column=Two }, { Row=Three; Column=Three }  //  1, 5 and 9rd position.
    { Row=One; Column=Three }, { Row=Two; Column=Two }, { Row=Three; Column=One }  //  3, 5 and 7th position.
  ]

let cells =
  List.ofSeq <|
    seq {
      for row in [One; Two; Three] do // Goes thorugh the row
      for column in [One; Two; Three] do // Goes through the column
      yield { Row=row; Column=column }  //yield adds a single item into the sequence being built (similar to the "
    }
```

Figure 12. Funciones waysToWin y cells

La función waysToWin contiene las ocho combinaciones posibles para ganar que ya se mencionaron anteriormente, dicha función compara las entradas que se tienen en el tablero, y si estas coinciden con alguna de las ocho combinaciones, entonces se procede a evaluar el carácter que cumple dicha coincidencia para determinar al ganador.

La función cells, como su nombre lo indica, es para las casillas, es decir, se recorren las columnas y las filas hasta llegar a la posición que se ha determinado por el jugador en turno.

Se puede observar que las casillas están determinadas por “one, two, three”, y no por “1, 2 y 3”. Esto es importante, pues más adelante se mencionará el porqué de dicha asignación.

```
let modify f (board: Board) (position: Position) = // Function were the board is modified ac
    set (f (select board position)) board position

  let placePieceIfCan piece = modify (function | Unspecified -> Letter piece | x -> x) // Modi

let makeMove (board: Board) (move: Move) = // Allows to make the move according to the previ
    if select board move.At = Unspecified
    then Some <| placePieceIfCan move.Place board move.At
    else None
```

Figure 13. Funciones para los movimientos realizados

Con modify se realiza la selección de la posición con las funciones que ya se han mencionado anteriormente. Con placePieceIfCan se modifica el tablero, se puede observar como se asigna el carácter al valor Unspecified que se tenía anteriormente. Finalmente se realiza el cambio o “movimiento” con makeMove, pues ya se ha evaluado con las 2 anteriores funciones.

Se determina al ganador de la siguiente manera:

```
let ``map 3`` f (a, b, c) = f a, f b, f c // To verify if the 3-in-line characters matches to the "X" or

let winner (board: Board) = //when the winner is determined
    let winPaths = List.map (``map 3`` (select board)) waysToWin //it is compared with one of the ways to

    if List.contains (Letter X, Letter X, Letter X) winPaths // If the winning path contains only "X" then
    then Some X
    else if List.contains (Letter O, Letter O, Letter O) winPaths  // // If the winning path contains only
    then Some O
    else None // Otherwise, if all the cells are occuppied, then it is called a Draw

let slotsRemaining (board: Board) = //if there are cells left
    List.exists ((=) Unspecified << select board) cells // Unespecified is assigned to every cell left
```

Figure 14 Funciones winner y slotsRemaining

Con map 3 se puede verificar que los 3 caracteres en línea coincidan con alguna de las combinaciones de waysToWin(figura 12) en la función winner (figura 14). En esta última se evalúa que la lista contenga los 3 caracteres en línea X o bien, O. Para de esta forma nombrar al ganador, o de lo contrario un empate.

Con slotsRemaining, en el tablero se evalúa que, en efecto, exista dicha casilla, es decir que se encuentre dentro del rango (3), y posteriormente que tenga un valor no especificado, lo cual, indica que la casilla está vacía y se encuentra disponible.

A partir de ahora, se muestran las secciones de código que dan inicio al juego tanto para el usuario como para la computadora

```
// GAME START
let initialGameState = { Board=emptyBoard; WhoseTurn=X } //Prints the empty board and gives the first turn to the X (computer)

let parseOneThroughThree = function // Converts the numeric input to a letter in order to make the cell selection
    | "1" -> Some One
    | "2" -> Some Two
    | "3" -> Some Three
    | _ -> None
```

Figure 15. GameStart

initialGameState es la función que imprime el tablero vacío al inicio del juego y le cede el primer movimiento a X (computadora).

La función parseOneThroughThree es aquella a la que se hizo referencia anteriormente (figura 12), en ella se lleva a cabo el cambio o “parseo” en el tipo de entrada, pues al recibirse algún dígito o valor numérico (1, 2 o 3) este es

asignado a one, two o three según corresponda, o en su defecto a "- "que sería ninguno de los anteriores.

```
let parseMove (raw: string) =
  match raw.Split [|' '|] with // omitts the blank space in the input and cuts the string
    | [|r; c|] -> //Receives row and then column
      match parseOneThroughThree r, parseOneThroughThree c with // Parses the 1, 2 or 3
        | Some row, Some column -> Some { Row=row; Column=column } // Assings the number to each variable (row and column)
        | _ -> None
    | _ -> None
```

Figure 16. Función parseMove

Con la función parseMove, al igual que con la función anterior, se hace el "parseo" del movimiento, o, en otras palabras, de la entrada del jugador. Se tiene la función Split, que corta la cadena ingresada y la separa del espacio en blanco que se encuentra entre los números de las casillas seleccionadas, le asigna el primer dígito a r (row o fila) y el segundo dígito a c (columna o columna) y se hace el llamado a la función parseOneThroughThree para hacer la asignación en forma de palabra y continuar con la evaluación de casillas y condiciones que se han explicado anteriormente.

Con las siguientes funciones se da inicio a los movimientos de ambos jugadores.

```
let rand s e =
  System.Random(System.DateTime.Now.Millisecond).Next(s, e)  // Establishes the time the com

let easyAiVHuman = function
  | X -> sprintf "%d %d" (rand 1 4) (rand 1 4)  // Generates random position between 1 and 3
  | O -> System.Console.ReadLine ()  // Gets the player input for row and column

let consoleAi = { PlaceLetter=easyAiVHuman; PrintLine=printfn "%s" }

playIo initialGameState consoleAi // Call to the function of the GAME START
```

Figure 17. Inicio de los movimientos

La función rand genera el movimiento a ejecutarse por la computadora y establece el tiempo que le tomará elegir dicha jugada en base a la hora del sistema.

Con easyAiHuman, si el turno es de X, se imprime lo que ya se generó en rand, y si el turno le corresponde a O, se lee la casilla en la que desea colocar su movimiento el usuario, posteriormente se hace el llamado al resto de las funciones.

```
let rec readMoveIo letter =
  placeLetter letter >>= fun line ->
  match parseMove line with
    | Some position ->
      pureIo { At=position; Place=letter }
    | None ->
      printLine "Por favor, ingresa el número de fila y columna correctamente" >>= fun () ->
      readMoveIo letter

let rec nextMoveIo board letter =
  readMoveIo letter >>= fun move ->
  match makeMove board move with
    | Some newBoard -> pureIo newBoard
    | _ ->
      printLine "Casilla ocupada, ingresa otra posición" >>= fun () ->
      nextMoveIo board letter
```

Figure 18. Movimientos ingresados

En la imagen superior se muestran las funciones que se encargan de leer el movimiento y de asignar el movimiento al siguiente jugador, si se ingresa una casilla incorrecta que van fuera del rango se imprime el mensaje de error, y si se ingresa una casilla ocupada se le pide al jugador que ingrese otros valores hasta que sea aceptado el movimiento. Así hasta que se hayan llenado todas las casillas o haya un ganador con alguna de las ocho combinaciones posibles.

V. CONCLUSIÓN

La inteligencia artificial juega un papel de suma importancia en el área de las Tecnologías de la Información, pues actualmente tiene una inmensa cantidad de aplicaciones en distintas ramas de la ciencia, y en un futuro, indudablemente tendrá un mayor alcance con el constante crecimiento de la tecnología y de las necesidades del ser humano.

Tener la oportunidad de conocer un poco más sobre el tema e implementarlo a un juego aparentemente sencillo, sin duda es una buena manera de comenzar la inmersión en la IA.

Si bien se trata de Tic Tac Toe, un juego bastante sencillo y sin mayor complicación de comprensión, realizar su análisis y la estructura que llevan las reglas del juego, para a su vez analizar un algoritmo e identificar la manera de abordar ambos factores, resulta todo un reto, pues se pueden presentar demasiadas complicaciones que harán ver el juego como algo bastante complejo contrario a la idea que se tiene de él.

Cabe destacar, que el lenguaje de programación F# es por su parte algo complejo y relativamente nuevo, por lo que encontrar la documentación adecuada significa una gran inversión de tiempo y de investigación, lectura y comprensión, pues además de analizar el código se debe encontrar la manera de implementarlo en el lenguaje en cuestión.

Para concluir, en la realización de este proyecto todos los factores mencionados fueron muy importantes, desde la comprensión del juego y del algoritmo, de los conceptos aplicados, así como el análisis y la manera de interpretarlo, tanto para codificarlo como para explicarlo posteriormente. Sin duda alguna ha sido un reto muy interesante.

REFERENCIAS

[1] Bramer, Max, comp. Artificial intelligence: an international perspective. Berlin: Springer Verlag, 2009.

[2]Marín Morales, Roque, y José Tomás Palma Méndez. Inteligencia artificial: técnicas, métodos y aplicaciones. Madrid: McGraw-Hill, 2008.

[3] Plasencia, C. (2017, 19 septiembre). *El algoritmo Minimax y su aplicación en un juego*. DevCode Tutoriales. https://devcode.la/tutoriales/algoritmo-minimax/