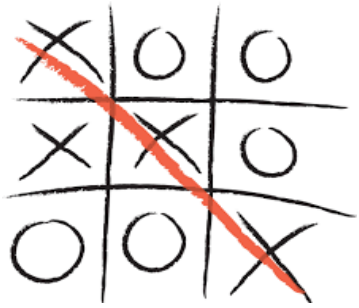
# 3 EN RAYA



# ALGORITMO MINIMAX

- Caso NO óptimo: árbol de profundidad 2 en cada jugada de MAX
- Caso óptimo: árbol completo en cada jugada de MAX

Nicolás Enrique Linares La Barba
2018/2019

# Índice

# 1 INTRODUCCIÓN

Este proyecto se ha centrado en el desarrollo del algoritmo minimax, aplicado en un juego conocido como es el 3 en raya o tic-tac-toe. En ciertos juegos, las decisiones que se van tomando son importantes para el desenlace del mismo, perjudicando o beneficiando a un jugador más que al otro. En los entornos multiagente, cualquier agente tiene que considerar las acciones de los otros agentes y como afectan a su propio bienestar.

La imprevisibilidad de estos agentes, pueden introducir muchas posibles contingencias en el proceso de resolución de problemas (entornos multiagente cooperativos y competitivos). Los entornos competitivos, en los cuales los objetivos de un componente están en conflicto, dan ocasión a problemas de búsqueda entre adversarios – a menudo conocidos como juegos.

En Inteligencia Artificial, ''los juegos'' son, por lo general, una clase especializada - que los teóricos de juegos llaman juegos de suma cero, de dos jugadores, por turnos, determinista, de información perfecta.

En nuestra terminología, significan entornos deterministas, totalmente observables con dos agentes cuyas acciones deben alternar y en los que los valores utilidad, al final de juego, son siempre iguales y opuestos (por ejemplo, si un jugador gana (+1), el otro jugador necesariamente pierde (-1)).

# 2 EXPLICACIÓN DEL ALGORITMO (caso NO óptimo)

## 2.1 Idea general

Shannon en 1950 propuso un cambio, que los programas deberían cortar la búsqueda antes y entonces aplicar una función de evaluación heurística a los estados, convirtiendo, efectivamente, los nodos no terminales en hojas terminales.

- Sustituir la función de utilidad por una función de evaluación heurística EVAL que estime la utilidad de la posición.
- Sustituir el test-terminal por un test-límite que decide cuando aplicar EVAL.

Una función de evaluación devuelve una estimación de la utilidad esperada de una posición, tal como hacen las funciones heurísticas que devuelven una estimación de la distancia al objetivo.

## 2.2 Creación del árbol

En este caso no se va a rellenar el árbol entero para tomar una decisión. Se va a simplificar esto, de manera que vamos a crear un árbol de profundidad 2. Es decir, MAX va a tomar una decisión según la posible respuesta de MIN. Esto convierte al algoritmo en no óptimo pues no comprueba todas las decisiones posibles, si no solo la reacción que puede tener MIN al movimiento realizado por MAX. Además, en estos nodos "terminales" se va a calcular la función heurística que luego va a ser comparada con la función minimax para escoger el nodo ganador.

h(n) = (nº filas, colum y diago libres para Max) - (nº filas, colum y diago libres para Min)

Observamos en la Fig. 1 la forma del árbol para cada jugada de MAX.

## 2.3 Aplicación del algoritmo Minimax para el caso NO óptimo

Como se ha visto en la Fig. 1 el árbol tiene una profundidad de 2, por lo que no es necesario hacer una función recursiva, con una iteración en cada hijo del nivel 1 ya se puede encontrar el mínimo valor para cada nodo. Finalmente, solo se itera en ese nivel 1 para escoger la jugada con más beneficio.

En el ejemplo de la Fig. 1 se puede ver como escoge el nodo rodeado con un círculo porque es el que mayor beneficio le proporciona.

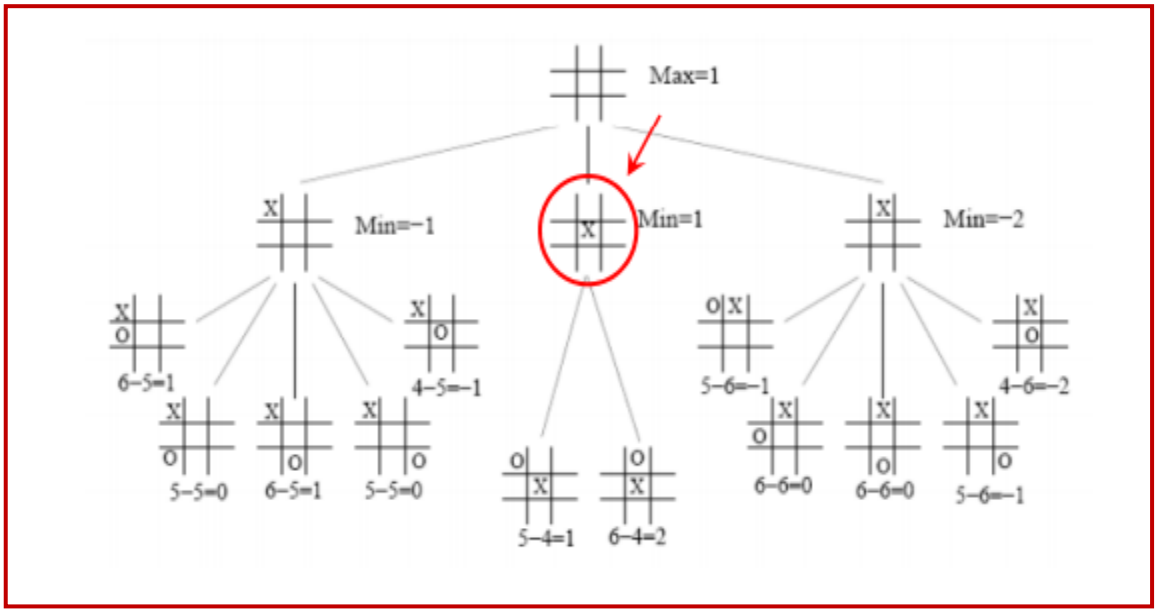



Figura 1.

# 3 EXPLICACIÓN DEL ALGORITMO (caso óptimo)

## 3.1 Idea general

Consideraremos juegos con dos jugadores (MAX y MIN). MAX mueve primero, y luego mueven por turno hasta que el juego se termina. Al final de juego, se conceden puntos al jugador ganador y penalizaciones al perdedor.

MAX → la máquina.

MIN → el ser humano.

El estado inicial y los movimientos legales para cada lado definen el árbol de juegos. El número sobre cada nodo hoja indica el valor de utilidad del estado terminal desde el punto de vista de un jugador (valores altos, buenos para MAX y malos para MIN). El trabajo de MAX es usar el árbol de búsqueda (en particular la utilidad de estados terminales) para determinar el mejor movimiento.

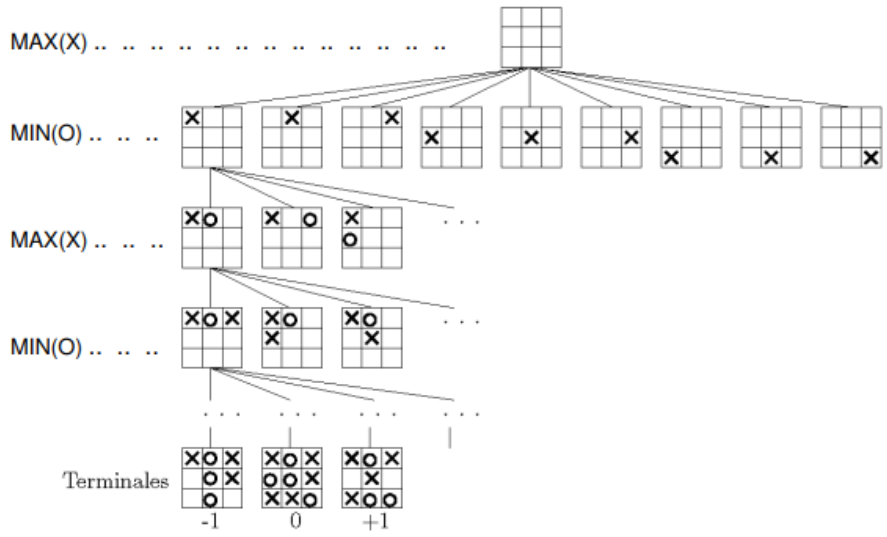
## 3.2 Creación del árbol

La partida puede ser empezada por cualquiera de los jugadores. Cuando el turno es de MAX debe coger el estado actual de la tabla 3x3 para comenzar a analizar las jugadas, de tal forma que va a realizar una predicción de todas las jugadas posibles (rellenando todo el árbol) para tomar el movimiento que más le beneficie. Procede de la siguiente manera.

Como se puede observar en la Fig. 2, MAX va a crear un árbol de decisiones a partir de la tabla actual, que se almacenará en la raíz del árbol. Como es el turno de MAX, comprueba cuantos huecos posibles tiene para marcar en la tabla y crea un nodo hijo con cada uno de los posibles movimientos. En el siguiente nivel del árbol, imita el turno de MIN con todos los huecos restantes. Así sucesivamente, hasta que llega a las jugadas terminales que completan el tablero.

Mientras se va a crear el árbol se comprueba si son jugadas ganadoras para algún agente, en su caso se valora el nodo con un número: si gana MIN es -1, si gana MAX es +1 y si empatan es 0.

Es importante destacar que: si a mitad del árbol un nodo es ganador o perdedor, se le asigna el valor +1 o -1, respectivamente, y se poda el árbol pues es inútil seguir por ahí.




Un árbol (parcial) de búsqueda para el juego de tic-tac-toe (tres en raya)

Figura 2.

## 3.3 Aplicación del algoritmo Minimax para el caso óptimo

Una vez creado el árbol se va a aplicar el algoritmo Minimax para escoger el movimiento de MAX. Es un algoritmo recursivo que comienza por el final del árbol.

En la Fig. 2 se puede ver como los nodos de abajo del todo (los terminales) tienen un valor asociado, si gana MIN es -1, si gana MAX es +1 y si empatan es 0. Este valor es el que se devolverá con las funciones recursivas de MAX-VALOR y de MIN-VALOR, cuando sea un nodo terminal.

MAX-VALOR → busca maximizar.

MIN-VALOR → busca minimizar.

Esto significa que MAX-VALOR va a escoger el nodo con mayor valor de beneficio y MIN-VALOR va a escoger el nodo con mínimo valor de beneficio. Este valor de beneficio se irá actualizando a medida que se sube en el árbol y finalmente se cogerá el nodo con mayor beneficio para la máquina.