

Juan Pablo Duque Ordoñez

Luis Alfonso García Waldron

**Informe de Proyecto I+D de Algoritmia y Programación II**

**Triqui Inteligente**

Algoritmia y Programación II

Mayra Zurbaran Nucci

Barranquilla, 5 de Octubre de 2015

**Introducción**

El triqui es un juego de papel y lápiz entre 2 jugadores, X y O, que marcan los espacios de un tablero 3x3 alternadamente, cuyo objetivo es lograr poner 3 de los símbolos del jugador en línea, esta línea puede ser horizontal, vertical o diagonal. El juego termina cuando alguno de los jugadores haya ganado o no haya más jugadas permitidas (Empate).

Ahora, para hacer un poco de análisis matemático del juego, tenemos que la cantidad de estados de juego, en otras palabras, la cantidad de posibles posiciones de juego, equivale a 9!, es decir, 9\*8\*7\*6\*5\*4\*3\*2\*1 = 362880, sin embargo, hay muchos estados de juego prohibidos o que son imposibles, por lo que las posibilidades se reducen a 278538[1] como lo muestra esta tabla:

|  |  |
| --- | --- |
| Profundidad | Estados |
| 1 | 23 |
| 2 | 162 |
| 3 | 929 |
| 4 | 4753 |
| 5 | 20900 |
| 6 | 75434 |
| 7 | 114914 |
| 8 | 214983 |
| 9 | 278538 |

Para una persona normal puede parecer exagerado tantos estados, pero para una maquina procesar 278538 datos es cuestión de milisegundos.

**El Juego**

Nuestro juego triqui fue creado en Windows Forms usando el lenguaje C++, para el juego se utilizaron los siguientes controles de la interfaz de usuario, o UI en inglés (User Interface):

**- Label**: Para las etiquetas, mostrar los puntajes y texto plano.

**- Button:** Para la navegación en el Form y elegir el modo de juego y dificultad, así como para reiniciar el tablero.

**- DataGridView:** Para mostrar el tablero 3x3 y realizar ahí el juego.

Para la interacción con la interfaz de usuario se utilizaron los siguientes eventos:

**- Click:** Para capturar los clicks en los botones.

**- CellClick:** Para capturar un click en las celdas del DataGridView y hacer más fácil la interacción con el tablero, ya que dicho evento captura las coordenadas de la celda del DataGridView que ha sido clickeada.

El juego permite 2 opciones de juego, 2 jugadores y VS AI (VS Artificial Intelligence o Inteligencia Artificial). El modo de 2 jugadores permite jugar a 2 jugadores humanos intercambiando turnos de manera automática. A su vez, el modo VS AI tiene 2 dificultades, Fácil e Imposible. En este modo de juego, un usuario humano juega contra la máquina, en el modo fácil es posible ganarle haciendo una estrategia de juego, pero en el modo imposible no hay manera posible de ganar, el porqué se explicará más adelante.

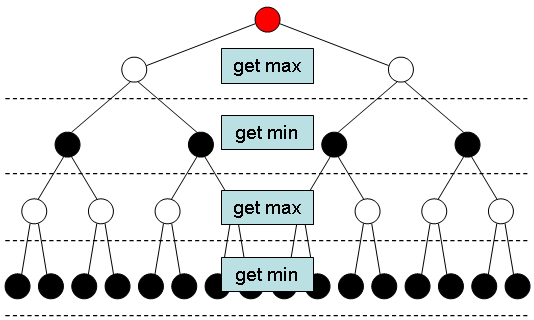
Las victorias en cada modo de juego se contabilizar en 2 contadores, y se reinician cada que se sale al menú principal y se inicia un nuevo juego en cualquier modo. El botón reiniciar limpia el tablero y devuelve los valores de juego a los predeterminados.

**El Algoritmo Minimax**

Para la programación de la lógica de juego de la máquina, fue necesaria cierta investigación en temas como Backtracking y recursividad, más específicamente, en un algoritmo llamado Minimax.

Backtracking se refiere a un algoritmo que es usado para encontrar todas las soluciones para un problema computacional y luego elegir una [2], su nombre se debe a que genera todas las posibles soluciones al problema planteado y luego regresa sobre sus pasos para dar una respuesta al problema original, por lo tanto es recursivo.

Minimax es un algoritmo recursivo que es capaz de elegir la opción más viable mediante el análisis de todas las posibles opciones y la maximización y minimización de estos valores de manera alternada.

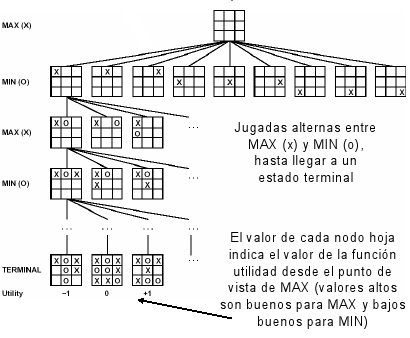


Se investigó sobre este algoritmo en diversas fuentes y se vio un video del MIT OpenCourseWare en YouTube titulado “6. Search: Games, Minimax, and Alpha-Beta”, parte de un curso de 2010 dictado en el MIT en Estados Unidos.

Para explicar la implementación del algoritmo Minimax en el triqui debemos explicar el sistema de cómo se analizan los estados de juego en el código.

El control DataGridView es una especie de matriz de 2 dimensiones, pero su manipulación es complicada, por lo que se usó un vector de 9 espacios (iniciando desde 0) como variable global donde se guardaba el estado actual del juego, así fue más fácil manipular el estado actual del juego y su validación. Para determinar al ganador se usó una función que retorna 1 si gana X y -1 si gana O, y para el empate se utilizó una función que retorna 0 si hay empate y 2 si no lo hay.

Como se dijo antes, Minimax minimiza un valor y maximiza otro, en este caso consideremos 3 casos base para la función Minimax, retorno 1 cuando gana X, retorno 0 cuando hay empate y retorno -1 cuando gana O, por lo tanto mientras más alto sea el valor retornado, más posibilidades tiene X de ganar o empatar y a su vez mientras el valor sea menor, más posibilidades tiene O. Esto quiere decir que nuestro jugador a maximizar es X y a minimizar es O.

El algoritmo se ejecuta luego de que juega el usuario y procede a analizar cada jugada posible en el estado actual de juego haciendo uso de llamadas recursivas hasta llegar al nivel más bajo del árbol donde se puede identificar un ganador o un empate en el estado de juego, en ese momento el algoritmo vuelve sobre sus pasos y selecciona el valor mínimo para la máquina en las jugadas de la máquina y el valor máximo para el usuario en las jugadas del usuario para asegurarse que el camino tomado nunca sea un camino en que el usuario pueda ganar.

Al implementar este algoritmo nos aseguramos que la máquina siempre gane o empate, ya que es capaz de ver todos los movimientos futuros y seleccionar cualquiera en que el usuario no pueda ganar.

Aquí está la implementación del algoritmo Minimax:

int minimax(String^ player) { //Toma como parametro un jugador X u O

int i, res;

int move = -1;

int min = 2, max = -2;

if (winner(Pos) == 1) { return 1; }; //Gana X

if (winner(Pos) == -1) { return -1; }; //Gana O

if (draw() == 0) { return 0; }; //Empate

for (i = 0; i < 9; i++) { //Para cada posicion del vector

if (player->Equals(jugador)) { //maximiza para X

if (Pos[i]->Equals(" ")) { //Si encuentra una posicion vacia

Pos[i] = jugador; //Asigna el simbolo del jugador

res = minimax(maquina); //Hace un llamado recursivo

Pos[i] = " "; //vuelve el vector a su estado original

if (res > max) { //Maximiza

max = res; //modifica el valor Maximo

move = i;

}

}

}

if (player->Equals(maquina)) { //minimiza para O y hace lo mismo

if (Pos[i]->Equals(" ")) {

Pos[i] = maquina;

res = minimax(jugador);

Pos[i] = " ";

if (res < min) { //Minimiza

min = res; //modifica el valor mínimo

move = i;

}

}

}

}

if (player->Equals(maquina)) { return min; }//retorna un mínimo para O

if (player->Equals(jugador)) { return max; }//retorna un máximo para X

}

**Inteligencia Artificial**

La máquina tiene 3 formas de jugar dependiendo del tablero, para el nivel Imposible, la maquina siempre jugará al centro si está libre, si esto no es posible analiza si puede ganar, perder o empatar en el siguiente turno y juega en esa posición. Finalmente, si no puede realizar nada de esto, para el nivel Imposible implementa el algoritmo Minimax para asegurarse que la posición elegida sea la más óptima y tener seguridad de que no va a perder, para el nivel fácil en cambio, no se implementa el algoritmo Minimax, sino que juega al primer espacio libre que encuentre, sea óptimo o no.

Para implementar el minimax en el turno de la máquina se analizaron todas las posiciones libres y se colocó el símbolo O en cada una de ellas, posteriormente se llamó a la función minimax con el parámetro jugador (que equivale a “X”), para analizar cada jugada posterior y obtener el valor máximo de esa posición. Finalmente, se elige la opción óptima mediante la minimización de los valores retornados en cada posición de esta forma:

for (i = 0; i < 9; i++) { //Para todos los elementos del vector

if (Pos[i]->Equals(" ")) { //analiza si hay una casilla libre

Pos[i] = maquina; //mueve a esa posicion

resTemp = minimax(jugador); //analiza los siguientes movimientos

if (resTemp < minimo) { //minimiza para O

minimo = resTemp;

movimiento = i; //Selecciona la posicion donde sea menor

}

Pos[i] = " "; //Vuelve a dejar el estado original del tablero

}

}

return movimiento; //Retorna el movimiento

Finalmente, el algoritmo pasa una posición donde va a hacer su movimiento y pasa los valores del vector Pos[] al DataGridView que se está utilizando para dibujar el tablero y representar ahí las X y O. Esto se realiza mediante una subrutina que analiza los elementos de Pos y los asigna a su respectiva casilla en el DataGridView.

**Referencias.**

[1] A. van Cranenburgh, R. Smid and M. van Someren, '"Tic-Tac-Toe".

[2] P.W. Purdom, '"Search rearrangement backtracking and polynomial average time," *Artif.Intell.*, vol. 21, no. 1, pp. 117-133.