Universidad Nacional Autónoma de México
Instituto de Investigación en Matemáticas Aplicadas y Sistemas
Posgrado en Ciencia e Ingeniería de la Computación
Inteligencia Artificial

- Fermin Cristobal Perez
- Braulio Ramses Hernandez Martinez
- Artemio Baños Gallardo
- Gerardo Mejia
- Bryant Sandoval

26 oct 2023

Tema: Aplicación de Algoritmo Minimax para el juego del Gato

El algoritmo desarrollado por nuestro equipo consta de los siguientes predicados:

Predicado	Explicación
victoria/2	Define un predicado que nos dice si se ha ganado tomando en cuenta las 8 formas de victoria en el juego del gato
iguales/4	Unifica todos sus argumentos devolviendo verdadero si corresponde a una de las formas de victoria en el gato
empate/1	Evalúa que el tablero haya sido agotado en cuanto a jugadas, es decir no hay números en la lista solo O (círculo) o X (equis). Eso es un empate
actualizar/4	Recursivamente busca unificar la jugada indicada con la cabeza (1, 2, 3 al 9, pues hay 9 posiciones en el tablero) en el tablero y reemplazandola la posición con una O o X al encontrarlo
imprimir_linea/3	Muestra en consola los valores de los argumentos pasados al predicado dándoles formato para dibujar una matriz
imprimir_tablero/1	Muestra en consola el tablero en forma de matriz cuadrada
fin/1	Se asegura si el juego ha sido concluido en tal jugada
siguiente/2	Cambia el turno al siguiente jugador
turno/3	Solicita el turno al jugador actual y actualiza el tablero. Si el jugador escribe ‘salir’ termina el juego

elimina/3	Elimina un valor de una lista. Para este proyecto queremos eliminar las jugadas para obtener las posiciones disponibles.
posDisponibles/2	Obtiene las posiciones disponibles en el tablero. Ver elimina
swap/2	Wrapper the predicado unir/3
unir/3	Sea una lista [Cabeza|Cola] pone la Cabeza al final de la lista [Cola|Cabeza]
ultimo/2	Responde si el elemento indicado es el último en la lista
ramas/3	Obtiene las posibles ramas a partir del número de posiciones disponibles

¿Cómo funciona?

Lenguaje Prolog.
En particular utilizando el lenguaje de programación prolog se puede implementar el algoritmo minimax para jugar tic tac toe, o el gato.
En primera instancia se implementa el mapeo del tablero sobre el que se juega, para ello

```
victoria([X1,X2,X3,X4,X5,X6,X7,X8,X9], P):-
    iguales(X1,X2,X3,P); iguales(X4,X5,X6,P); iguales(X7,X8,X9,P);
    iguales(X1,X4,X7,P); iguales(X2,X5,X8,P); iguales(X3,X6,X9,P);
    iguales(X1,X5,X9,P); iguales(X3,X5,X7,P).
iguales(X,X,X,X).
empate(Tablero):-
    \+ (member(1, Tablero);member(2, Tablero);member(3, Tablero);member(4, Tablero);member(5, Tablero);
    member(6, Tablero);member(7, Tablero);member(8, Tablero);member(9, Tablero)).
```

son los predicados “victoria”, ”iguales” y “empate”, estos generan las situaciones específicas de gane o empate según se presente la situación del Tablero actual.
Por otro lado, es necesario el predicado “actualiza” para reflejar las modificaciones hechas

```
actualizar(_,_,[],[]).
actualizar(Pos,Ficha,[Pos|Xs],[Ficha|Ys]):- actualizar(Pos,Ficha,Xs,Ys).
actualizar(Pos,Ficha,[X|Xs],[X|Ys]):- actualizar(Pos,Ficha,Xs,Ys),!.
```

por el usuario, o bien la computadora.

Ahora bien, para decidir el turno de la computadora se utiliza minimax implementado como un predicado.

```
minimax(Tablero, TableroSig):-
    write('Turno de la computadora.'),nl,
    posDisponibles(Tablero, Disponibles),
    ultimo(Disponibles, Last),
    ramas(Disponibles, Last, Ramas),
    max(Ramas, Tablero, (Pos,Val)),
    actualizar(Pos,o,Tablero,TableroSig), !.
```

Donde se consideran primero las posiciones disponibles para colocar un “o”, después se consideran las ramas que generan las posiciones disponibles, estas son calculadas con el siguiente predicado.

```
ramas([X|T], X, [Ramas]):- swap([X|T], Ramas), !.
ramas([X|T], Ultimo, Ramas):-
    X \= Ultimo,
    swap([X|T], C1),
    ramas(C1, Ultimo,R1),
    append(R1,[C1],Ramas).
```

El resultado de este predicado es una lista de listas que contienen las posiciones con distintas combinaciones de estas mismas, es decir, si en un momento se tiene un tablero cuyas posiciones disponibles son [4,5,6], ramas devolverá una lista con elementos como [4,5,6], [5,4,6], [4,6,5], etc. esto para simular los nodos hermanos que genera el supuesto árbol de minimax.

Siguiendo con las llamadas de minimax se tiene el predicado “max” con el que se comienza la decisión para colocar una ficha “o”, este mismo llama al predicado “min” que tiene un comportamiento análogo a max. A continuación la implementación de ambos predicados.

```
98  max([[X|Y]], Tablero, (Pos, Val)):-
99      actualizar(X,x, Tablero,TableroSig),
100     (
101         (
102             victoria(TableroSig,x), Pos = X, Val = -1;
103             empate(TableroSig), Pos = X, Val = 0
104         );
105         ultimo(Y,U),
106         ramas(Y,U,RamasY),
107         min(RamasY, TableroSig, (Pos,Val))
108     ),!.
109
110 max([[X|Y]|T],Tablero,(Pos,Val)):-
111     actualizar(X,x, Tablero,TableroSig),        %Genera un tablero con una posicion tentativa
112     (
113         (
114             victoria(TableroSig,x), Pos = X, Val = -1;  %Se da un valor para cada tablero final
115             empate(TableroSig), Pos = X, Val = 0
116         );
117         ultimo(Y, U),
118         ramas(Y,U,RamasY),                  %Calculan el resto de posiciones segun el movimiento tentativo
119         min(RamasY, TableroSig, (PosX,ValX))
120     ),
121     (
122         (ValX == 1, Val = ValX, Pos = PosX);  %Se decide si la posición tentativa es o no llega a un resultado deseado para
            min
123         max(T, Tablero, (Pos,Val)) %Se llama a Max para revisar el siguente nivel
124     ),!.
```

Así la computadora puede considerar que posición jugar dentro del tablero utilizando el algoritmo de minimax regresando un tablero actualizado.

```
min([[X|Y]], Tablero, (Pos, Val)):-
    actualizar(X,o, Tablero,TableroSig),
    (
        (
            victoria(TableroSig,o), Pos = X, Val = 1;
            empate(TableroSig), Pos = X, Val = 0
        );
        ultimo(Y,U),
        ramas(Y,U,RamasY),
        max(RamasY, TableroSig, (Pos,Val))
    ),!.

min([[X|Y]|T],Tablero,(Pos,Val)):-
    actualizar(X,o, Tablero,TableroSig),    %Genera un tablero con una posicion tentativa
    (
        (
            victoria(TableroSig,o), Pos = X, Val = 1; %Se da un valor para cada tablero final
            empate(TableroSig), Pos = X, Val = 0
        );
        ultimo(Y, U),
        ramas(Y,U,RamasY),                    %Calculan el resto de posiciones segun el movimiento tentativo
        max(RamasY, TableroSig, (Pos0,Val0))
    ),
    (
        (Val0 == -1, Val = Val0, Pos = Pos0);  %Se decide si la posición tentativa es o no llega a un resultado deseado
        para min
        min(T, Tablero, (Pos,Val))  %Se llama a Max para revisar el siguente nivel
    ),!.
```

¿Cómo implementamos minimax?

El predicado ramas es clave para darnos las diferentes permutaciones de ramas, e. g.

[[1, 2, 3] [3, 1, 2] [2, 3, 1]]

A partir de esta lista de listas max actualizará la Tabla Siguiente.

Después preguntará si en esta jugada se ganó (poda, es decir por lo menos habrá más victoria o empates después de esto pues es el mejor caso de decisión para la computadora, no evalúa derrotas) y devolverá el control a la recursión anterior.
En caso de que no haya victoría o empate se sigue actualizando con los hijos de este nivel

[[2, 3] [3, 2]]

El predicado max toma esta lista de ramas y vuelve a validar si hubo victoria, si hubo victoria entonces devuelve el control a la llamada recursiva superior hasta llegar a la inicial donde se tomará el valor de Pos como mejor valor, de otra manera seguirá buscando hasta encontrar Victoria o Empate.

Implementación Java

La implementación de java considera al tablero como una matriz de caracteres la cual será llenada por el usuario y la máquina conforme avanza la partida. Inicialmente se prepara el tablero con el método inicializar(), el cuàl llena el tablero de espacios en blanco.

```
public static void inicializar() { //Se llena de espacios vacíos el tablero.
    tablero=new char[3][3];
    for(int i=0; i<tablero.length; ++i) {
        for(int j=0; j<tablero.length; ++j) {
            tablero[i][j]=' ';
        }
    }
}
```

Luego se inicia el juego con el método iniciar() el cual se encarga de mantener la partida en ejecución mientras no haya un ganador ni haya un empate (el tablero aùn tenga espacios para jugar).

```
public static void juego(char[][] x) { //Inicia el juego por turnos y verificando que se puede seguir con el juego.
    turno=true;
    imprimeMatriz(x);
    while(true) {
        if(ganador(x)==0 && !turno && !estaLleno(x)) {
            turnoUsuario();
        }
        if(ganador(x)==0 && turno && !estaLleno(x)) {
            turnoCompu(x);
        }
        imprimeMatriz(x);
    }
}
```

El turno del usuario consiste en pedirle a este indique la fila y columna del tablero en la que desea colocar su ficha (la ficha del usuario en la implementación de Java es la o). Aquí el programa tiene cuidado de que la posición indicada por el usuario sea válida.

```
public static void turnoUsuario() {
    Scanner entrada = new Scanner(System.in);
    int fil,col;
    boolean vacio=true;
    while(vacio) { //No se sale del ciclo hasta que el usuario ponga coord. válidas.
        System.out.println("Ingresa la fila donde quieres poner tu ficha: ");
        fil=entrada.nextInt()-1;
        System.out.println("Ingrese la columna donde quieres poner tu ficha: ");
        col=entrada.nextInt()-1;
        if(fil<3 && fil>-1 && col<3 && col>-1) { //Comprueba que las coord. ingresadas sean válidas.
            if(tablero[fil][col]==' ') { //Checa si se puede poner ahí una ficha.
                tablero[fil][col]='O';
                vacio=false;
            }

            else {
                System.out.println("Por favor ingrese una posición vacía.");
            }
        }
        else
            System.out.println("Por favor ingrese una posición que exista en el tablero.");
    }
    turno=true;
}
```

El turno de la computadora consiste en decidir la posición a jugar utilizando el algoritmo minimax() con su versión implementando la poda alpha-beta. Dentro del código anexado a este trabajo se puede revisar màs a detalle la implementación hecha, este archivo contiene comentarios detallados que facilitan el entendimiento del código.

```
public static void turnoCompu(char[][] x) {
    int filChida=10,filTemp,colChida=10,colTemp,m,actual, alfa, beta;
    m=-9999;//m, alfa, beta toman valores grandes para forzarlos a cambiar de valor más adelante.
    alfa=-9999;
    beta=9999;
    char fanta;
    for(int i=0; i<limite; ++i) { //El ciclo for anidado recorre todas las posibles opciones que tiene la computadora derivadas del edo. inicial
        for(int j=0; j<limite; ++j) {
            if(x[i][j]==' ') { //Revisa si hay un espacio vacío para contemplar esa opción
                fanta=x[i][j];//Almacena el valor de esa pos. para después
                x[i][j]='X';//Agrega el posible escenario al tablero para seguir haciendo el árbol de búsqueda
                filTemp=i;
                colTemp=j;
                actual = min(x,0,alfa,beta);//Baja un nivel en el arbol a buscar el min
                x[i][j]=fanta;//Regresa al tablero al estado inicial
                if(m<actual) { //Revisa si ha encontrado una posición que tenga mejor resultado que una posición anterior.
                    m=actual;
                    filChida=filTemp;
                    colChida=colTemp;
                }
            }

        }

    }
    x[filChida][colChida]='X';//Coloca la ficha en la posición más conveniente
    turno=false;
}
```

Los turnos de los jugadores se iteran hasta que el juego termine, es decir, se encuentre un ganador o se genere un empate.