

Inteligencia Artificial

Práctica 5: Minimax

Verónica Esther Arriola Ríos
Pedro Rodríguez Zarazúa
Luis Alfredo Lizárraga Santos

Fecha de entrega: Miércoles 23 de Marzo de 2016

1. Objetivo

Conocer qué es un juego de suma-cero, cómo se define un juego (desde el punto de vista de teoría de juegos) para comprender el funcionamiento del algoritmo minimax como regla de decisión suficientemente buena.

2. Introducción

2.1. Teoría de juegos

Un juego puede definirse formalmente como una clase de problemas de búsqueda con los siguientes componentes:

- **Estado inicial.** Incluye la posición del tablero de juego e identifica al jugador que mueve.
- **Función sucesor.** Devuelve una lista de pares (movimiento, estado), indicando un movimiento legal y el estado resultante.
- **Test terminal.** Determina cuándo se termina el juego. Los estados donde el juego se ha terminado se les llama estados terminales.
- **Función de utilidad (función objetivo).** Asigna un valor numérico a los estados terminales. En el ajedrez o un juego de gato, el resultado es un triunfo, pérdida o empate, con valores +1, -1 ó 0. Algunos juegos tienen una variedad más amplia de resultados posibles (por ejemplo el backgammon).

2.2. Juegos de suma-cero

La propiedad de suma-cero significa que cuando un jugador gana el otro pierde. Es por eso que los juegos de suma-cero son un ejemplo de juegos de suma constante donde la suma de cada resultado es siempre cero. Como ya se mencionó, el juego del gato y el ajedrez son ejemplos de juegos de dos jugadores de suma-cero.

En teoría de juegos, una estrategia minimax es parte de una solución a los juegos de suma-cero.

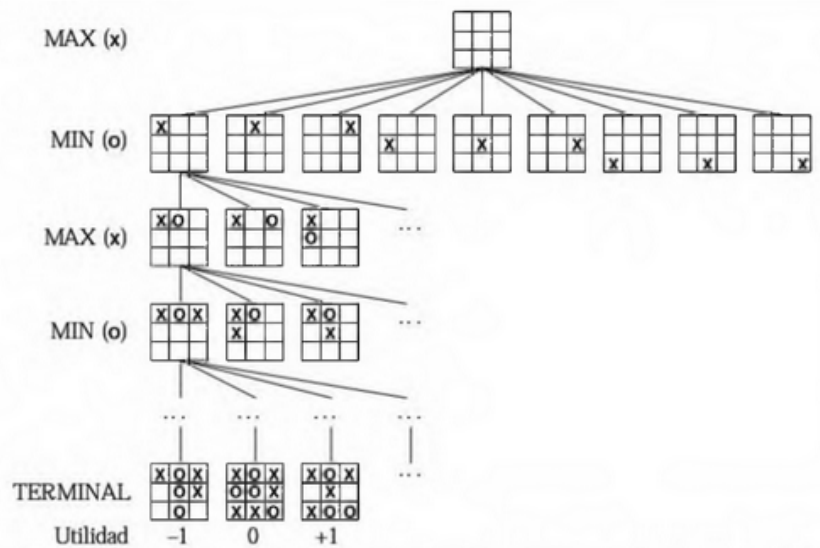


Figura 1: Árbol parcial de búsqueda (espacio de estados) para el juego del Gato. El primer jugador "X" intenta maximizar o ganar en su turno mientras intenta minimizar o hacer perder al jugador "O" en su respectivo turno. Los nodos finales tienen asignado un valor según la función de utilidad.

2.3. Teorema minimax

El teorema minimax dice:

Para cada dos-personas, un juego de suma-cero con estrategias finitas, existe un valor V y una estrategia para cada jugador tal que Dada la estrategia del jugador 2, el mejor resultado posible para el jugador 1 es V Dada la estrategia del jugador 1, el mejor resultado posible para el jugador 2 es $-V$.

El nombre minimax surge porque cada jugador minimiza el resultado máximo posible de su oponente. Como es un juego de suma-cero, él también minimiza su máxima pérdida.

2.4. Algoritmo

El algoritmo minimax es un algoritmo recursivo para elegir el siguiente movimiento en un juego de 2-jugadores (puede extenderse a n-jugadores). Un valor numérico es asociado a cada posición del estado del juego y es determinada por una función de evaluación (función de utilidad) que indica qué tan favorable podría ser para el jugador alcanzar dicha posición.

La función de utilidad determina que para el jugador A que maximiza (max), la utilidad en los estados finales donde gana es igual al valor +1, mientras que para el oponente B que minimiza (min) la utilidad de los estados finales donde B gana es -1, y 0 en otro caso.

Una vez determinado el valor de utilidad en los estados finales, se calculan los demás estados tomando el máximo o mínimo de los valores de utilidad si se trata del turno del jugador max o mín, respectivamente.

$$\text{VALOR-MINIMAX}(n) = \begin{cases} \text{UTILIDAD}(n) & \text{si } n \text{ es un estado terminal} \\ \max_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MAX} \\ \min_{s \in \text{Sucesores}(n)} \text{VALOR-MINIMAX}(s) & \text{si } n \text{ es un estado MIN} \end{cases}$$

Figura 2: Función de evaluación para cada estado n del espacio de estados.

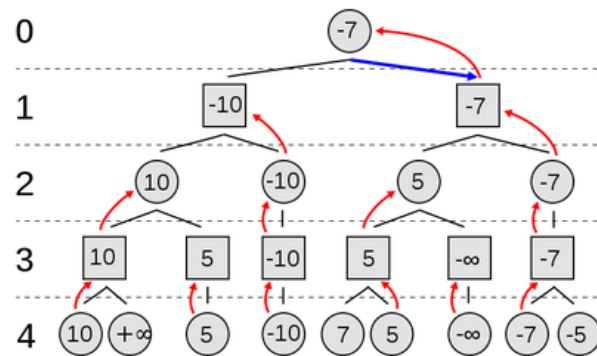


Figura 3: Ejemplo de un árbol con sus valores de utilidad. En altura par se busca el max de los hijos y en altura impar el min de los hijos. Las flechas rojas indican que la asignación se realiza desde las hojas hacia la raíz del árbol.

3. Implementación

Un algoritmo para la implementación puede ser:

```
funcion DECISION-MINIMAX(estado) devuelve una acción
    variables de entrada: estado //estado actual del juego

    v := ASIGNAR-VALOR(estado)
    return la acción de SUCESORES(estado) con valor v

funcion ASIGNAR-VALOR(estado) devuelve un valor utilidad
    if ES-FINAL(estado) then //utilidad de las hojas
        return UTILIDAD(estado)
    if ES-MAX(estado) then // utilidad del jugador "max"
        v := -infinito
        for s in SUCESORES(estado) do
            v := MAX(v, ASIGNAR-VALOR(s))
        return v
    if ES-MIN(estado) then // utilidad del jugador "min"
        v := infinito
        for s in SUCESORES(estado) do
            v := MIN(v, ASIGNAR-VALOR(s))
        return v
```

Figura 4

La función SUCESORES(*estado*) que se emplea recibe un estado y devuelve la lista de los siguientes estados al aplicar una acción legal en el juego.

4. Requisitos y resultados

Implementar el algoritmo minimax para el juego del gato, de tal manera que la llamada al método o función reciba un estado y devuelva la acción a tomar. Por ejemplo:

```

Terminal
File Edit View Search Terminal Help
rodrigo@rodrigo02 ~ $ lua -i minimax.lua
Lua 5.2.1 Copyright (C) 1994-2012 Lua.org, PUC-Rio
> estado:show()
X   X   -
-   0   0
-   -   -
> minimax(estado)
X   3   1
> ^C
rodrigo@rodrigo02 ~ $

```

Figura 5: Ejemplo de comportamiento de minimax en lenguaje de programación Lua. Se muestra el estado actual del juego del gato y el resultado de la función minimax es la acción correspondiente a tirar “X” en la posición (3,1) (esquina superior derecha).

Otra manera es que devuelva el siguiente estado en lugar de la acción a tomar. Por ejemplo:

```

Terminal
File Edit View Search Terminal Help
rodrigo@rodrigo02 ~ $ lua -i minimax.lua
Lua 5.2.1 Copyright (C) 1994-2012 Lua.org, PUC-Rio
> estado:show()
X   X   -
-   0   0
-   -   -
> minimax(estado)
X   X   X
-   0   0
-   -   -
> ^C
rodrigo@rodrigo02 ~ $

```

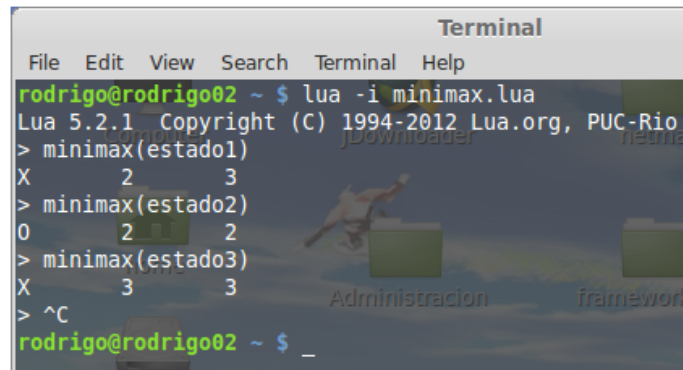
Figura 6: Implementación alternativa a la anterior donde se devuelve el estado siguiente en lugar de la acción a realizar en el estado actual.

Una vez que implementen el algoritmo Minimax, prueben y muestren los resultados obtenidos de su programa con los siguientes estados:



Figura 7

Ejemplo de resultado final del programa:



```
Terminal
File Edit View Search Terminal Help
rodrigo@rodrigo02 ~ $ lua -i minimax.lua
Lua 5.2.1 Copyright (C) 1994-2012 Lua.org, PUC-Rio
> minimax(estado1)
X 2 3
> minimax(estado2)
O 2 2
> minimax(estado3)
X 3 3
> ^C
rodrigo@rodrigo02 ~ $ _
```

Figura 8: Acciones a tomar para cada uno de los tres estados según la implementación de minimax.

5. Notas adicionales.

La práctica es individual, anexas a su código un archivo readme.txt con su nombre completo, número de cuenta, número de la práctica y cualquier observación o notas adicionales (posibles errores, complicaciones, opiniones, críticas de la práctica o del laboratorio, cualquier comentario relativo a la práctica).

Pueden agregar cualquier biblioteca extra, sólo asegurense de que se encuentre bien comentada.

Compriman la práctica en un solo archivo (.zip, .rar, .tar.gz) con la siguiente estructura:

- ApellidoPaternoNombreNúmeroDepráctica.zip (por ejemplo: LizarragaLuis05.zip)
 - Minimax
 - src
 - ◇ Minimax.java
 - readme.txt

La práctica se entregará en la página del curso en la plataforma AVE Ciencias.

O por medio de correo electrónico a luislizarraga@ciencias.unam.mx con asunto Práctica05[IA 2016-2]

La fecha de entrega es hasta el día miércoles 23 de Marzo a las 23:59:59 hrs.

Para quienes hagan la práctica con netbeans, pongan el directorio del proyecto dentro de la carpeta src