NOMBRE: Alex Benavidez

MATERIA: Inteligencia Artificial

FECHA: 14/07/2020

ALGORITMO MINIMAX

El algoritmo de búsqueda Minimax, es una búsqueda de profundidad limitada. El nombre del algoritmo se da al considerar que dada una función estática que devuelve valores en relación al jugador maximizante, este trata de maximizar su valor mientras que el oponente trata de minimizarlo. En un árbol de juego donde los valores de la función estática, están en relación con el jugador maximizante, se maximiza y minimiza de forma alterna de un nivel a otro.

Teoria de Juegos

Minimax es un algoritmo de decisión para tratar de minimizar la perdida máxima, que se espera en los juegos que contienen adversarios. El funcionamiento de Minimax se define a como se puede elegir el mejor movimiento, dado el caso que el contrincante escoja el peor movimiento. Realizar el algoritmo de Minimax se realiza en juegos sencillos, ya que implementarlo completamente consumiría gran cantidad de tiempo y memoria. Según Claude en el artículo Programming a Computer for Playing Chess, propuso que se limitara la profundidad de la búsqueda en el árbol de posibilidades y determinar el valor mediante razonamiento heurístico, con la finalidad de optimizar el algoritmo se puede limitar ya sea la búsqueda por nivel de profundidad o por tiempo de ejecución, otra forma seria también utilizar la poda alfa-beta, esta forma de optimización se basa en suponer que un jugador contrario no permita que hagamos nuestros mejores movimientos.

Características:

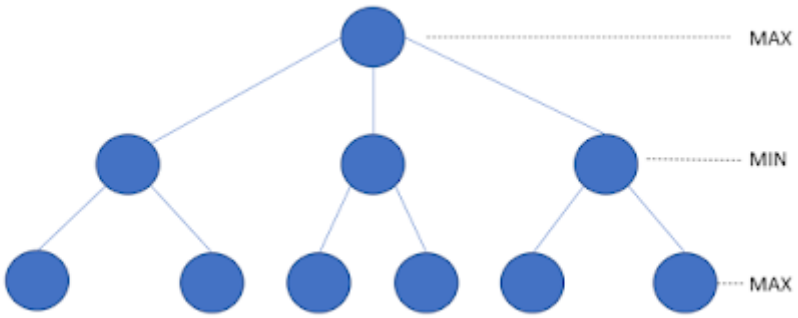
- La facilidad que dispone para crear situaciones complicadas utilizando reglas sencillas.
- Este algoritmo se puede utilizar con humanos a nivel de escalas.
- Los juegos que utilizan este algoritmo terminan siendo adictivos.
- El adversario introduce incertidumbre ya que no se conoce el movimiento que realizará.
- El algoritmo Minimax utiliza procedimientos recursivos, esto culmina cuando sucede las siguientes condiciones:
    1. Gana un jugador.
    2. La búsqueda a explorado N capas, y este es el límite que se ha establecido.
    3. El tiempo a culminado para la búsqueda.
    4. Se a llegado a la situación en donde ya no se producen grandes cambios al pasar de un nivel a otro.

Estructura utilizada en Minimax

In [8]:

```
from IPython.display import Image
Image(filename='estructura_minmax.PNG')
```

Out[8]:



Pasos que realiza el algoritmo MiniMax

- Generar el árbol de juego. Se deben generar todos los nodos hasta llegar a un estado meta o de llegada.
- Cálculo de los valores de la función de utilidad para cada nodo meta.
- Se calcula el valor de los nodos superiores a partir del valor de los nodos inferiores. Se eligen los valores mínimos y máximos, esto lo que hace es representar los movimientos del jugador y del adversario.
- Se elige la jugada tomando en cuenta los valores del nivel superior.

El algoritmo lo que hace es explorar los nodos del árbol y le asigna un valor numérico mediante la función de utilidad, empieza por los nodos terminales y sube hacia la raíz. La función de utilidad lo que hace es definir si la posición es buena para un jugador. En un juego se obtiene:

- Posición inicial
- Conjunto de operadores esto define las movidas.
- Estado meta o final
- Funciona de utilidad (gana, pierde, empata) , las reglas del juego determina los movimientos que se pueden dar.

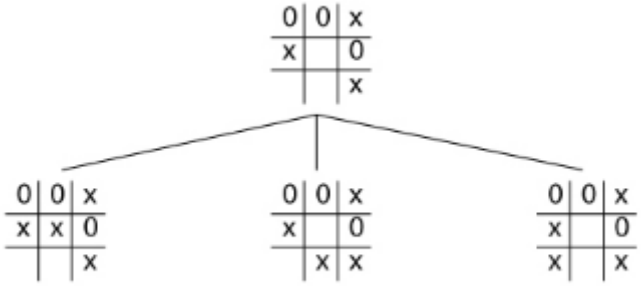
Ejemplo práctico el funcionamiento del algoritmo(Desarrollo de un tres en raya)

En el algoritmo Minimax el espacio de búsqueda queda definido por:

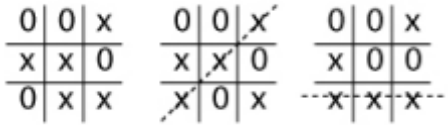
1. Estado Inicial: Configuracion Inicial del juego

0	0	x
x		0
		x

2. Operadores: Jugadas legales que se pueden hacer en el juego, en este caso no se puede marcar una casilla que haya sido marcada.

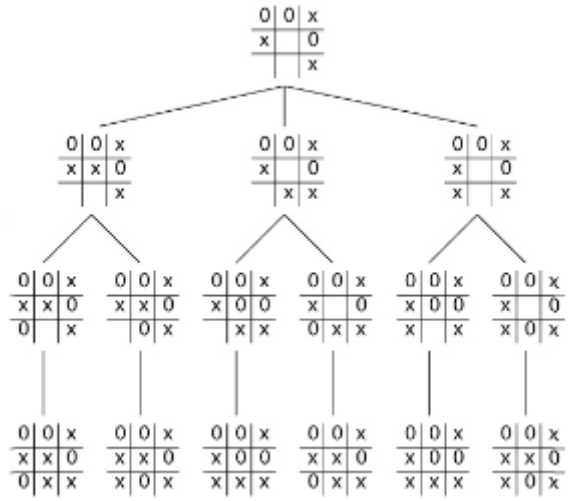


3. Condicion Terminal: Permite saber cuando se acabo el juego, en este caso se acaba el juego cuando un jugador marca tres casillas que sean legales, ya sea horizontalmente, verticalmente o en diagonal, finalmente quedan empate si estan marcadas todas las casillas.

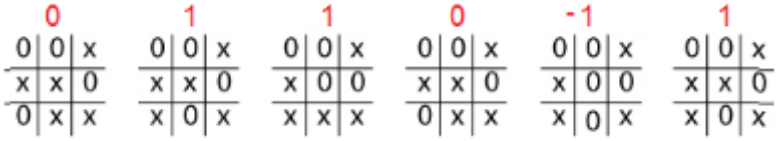


4. Implementacion de Minimax:

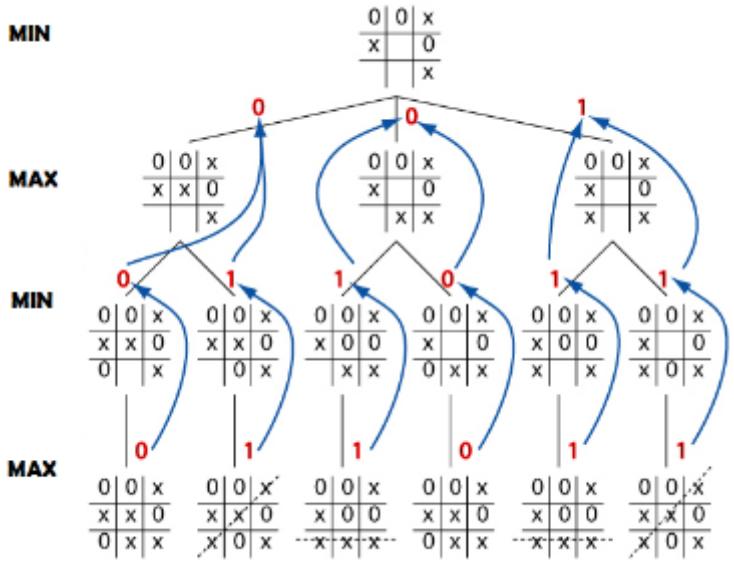
- Primero genera un arbol de soluciones completo a partir de un nodo dado.



- Para cada uno de los nodos finales se pricede a buscar la funcion de utilidad de estos. En el ejemplo que se muestra se usa el numero 0 pra partidas que queden en empate, 1 para los que gane la maquina y finalmente un -1 si gana el jugador humano.



- Lo que hara el algoritmo Minimax al momento se este vaya retrocediendo, es comunicarle a la llamada recursiva superior cual es el mejor nodo nodo hoja alcanzado hasta el momento. En cada llamada recursiva tiene que saber a quien le toca jugar, para asi poder analizar si el movimiento fue implmentado por la maquina o por el jugador humano, ya que cuando sea el turno de la maquina se pretende maximizar el resultado, y cuando sea el turno de la persona minimizar su resultado.



- Al finalizar el algoritmo nos devolvera la jugada que debe realizar la maquina para maximizar sus posibilidades y poder bloquear las posibilidades del rival

Alpha-Beta prunning

Este se aplica a un algoritmo de Minimax estandar,permite eliminar todos los nodos que no afecten a la decision final. Es una tecnica de optimizacion para el algoritmo Minimax, debido a que reduce el tiempo en el cual se realiza el calculo. Esto permite de una manera buscar de manera mas rapida a ir a niveles de profundidad mucho mayor en el arbol de juego. Su eficiencia depende del orden en que se visitan los sucesores, en donde se examina primeramente los sucesores que sean mejores.

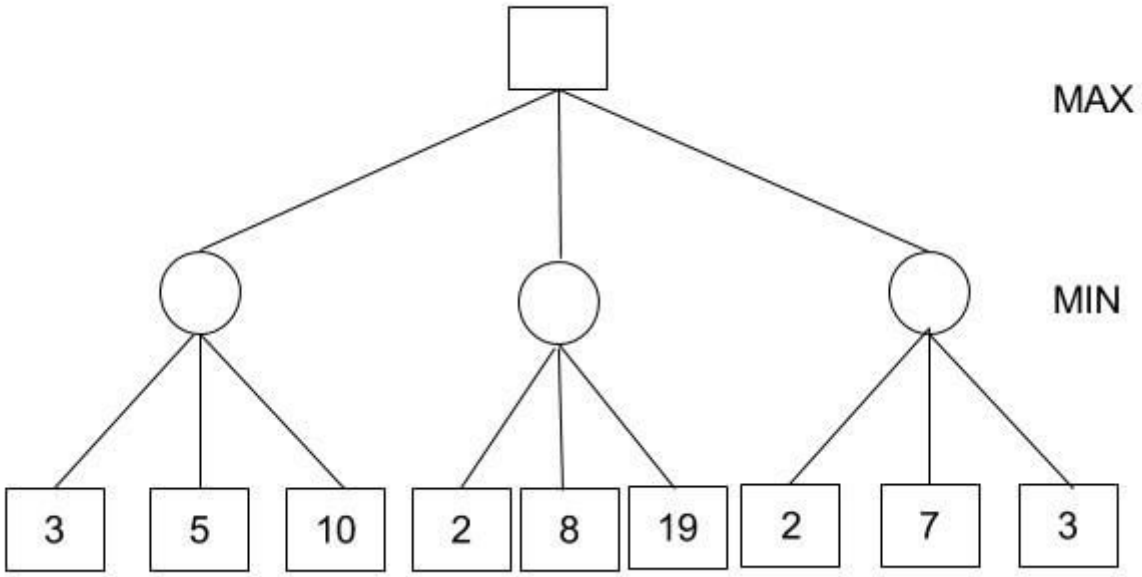
Características de Alpha-Beta prunning

- Mejora del Algoritmo Minimax; aplicado en juegos de adversarios por turnos
- Se aplica en espacios de estados demasiado grandes como para analizar todos los nodos
- La información es imperfecta; es decir, no se conoce el estado del contrincante. P. ejem. En juegos donde no se ve el tablero del adversario
- Produce la misma jugada que se obtendría con MiniMax, pero elimina todas las ramas que posiblemente no influirán en la decisión final.
- Alfa: valor de la mejor opción encontrada hasta entonces en un punto de elección a través de la ruta de MAX;
- Beta: el valor de la mejor (valor más bajo) opción encontrada hasta entonces en un punto de opción a lo largo de la ruta MIN.
- Conforme se efectúa la búsqueda Alfa-Beta se van actualizando los valores de alfa y beta y se poda
- Omitir la expansión de nodos que por sus valores no pueden ser los mejores (peores).
- Interrumpe la búsqueda en algún nivel y aplica evaluaciones heurísticas a las hojas (profundidad limitada)

- Alfa-Beta permite búsqueda dos veces más profunda.
- Ordenamiento de los operadores, resultante del conocimiento o experiencia.
- Únicamente importa el orden y no los valores exactos.
- La poda no afecta al resultado final.

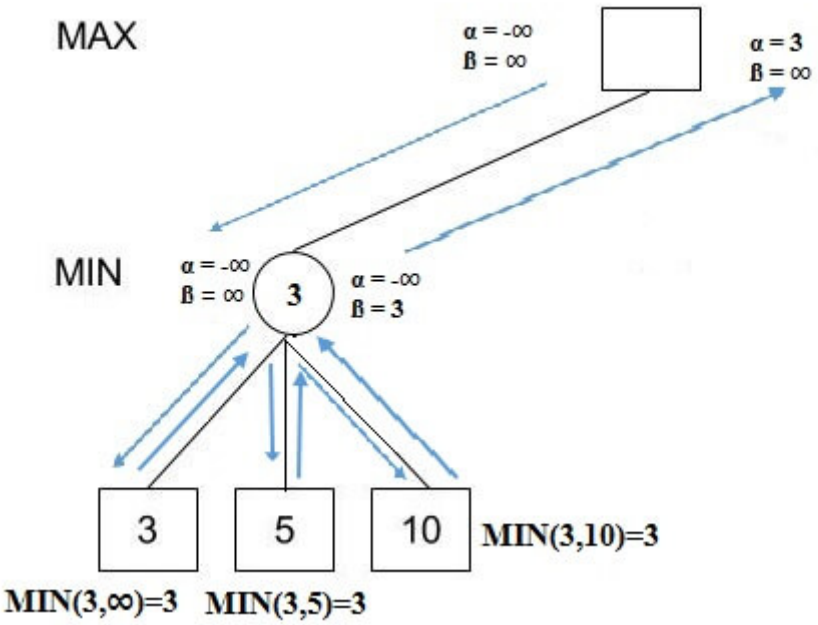
Ejemplo práctico el funcionamiento del algoritmo

1. Inicialice alpha = -infinity y beta = infinity como los peores casos posibles. La condición para podar un nodo es cuando alfa se vuelve mayor o igual que beta.



2. Se comienza asignando valores iniciales de alfa y beta en los nodos raiz y como alfa es menor que beta no procedemos a podarlo.

3. Se procede llevar los valores de alfa y betaa al nodo secuandario de la izquierda. Despues desde el valor de utilidad del estado del termianl se actualiza los valores de alfa y no actaulizamos el valor de beta. Otra vez no podamos ya que la condicion sigue siendo la misma. Despues se retrocede a la raiz y se establece alfa -3 ya que es el valor minimo que alfa puede tener.



4. En este caso tenemos alfa -3 y beta-infinito en la raiz, en tonces no podamos, calculando MIN-2, infinito, obtnemos alfa-3 y beta-2

5. En este caso Podamos el segundo y el tercer nodo secuandario debido a que alfa es ahora mayor que beta.

6. Alfa en la raiz sigue siendo 3 porque es mayor que 2. Se procede a llevar al nodo secundario mas a la derecha. Finalmente se actualiza beta a 2 y alfa sigue teniendo el valor de 3.

7. Se procede a podar el segundo y tercer nodo secundario ya que alfa es mayor que beta.

8. Se procede a obtener 3,2,2 en los nodos MIN izquierdo, central y derecho. Calculando MAX en 3,2,2 se obtiene 3, en donde se ya obtenido de manera correcta la decision minimax

CONCLUSIONES

- El algoritmo minimax permite utilizar la inteligencia artificial en los juegos lo que hace que los juegos sean mas interesantes, en donde nos facilita jugar contra nuestra propia maquina y esto nos hace mas complicado poder ganar.
- Alpha-Beta Prunning es una tecnica que mejora el algoritmo minimax, lo que permite que nuestra maquina tenga las mayores posibilidades de poner ganar el juego

BIBLIOGRAFIA

- http://algoritmojuegos.blogspot.com/2018/11/algoritmo-minimax_23.html (http://algoritmojuegos.blogspot.com/2018/11/algoritmo-minimax_23.html)

- http://iaupsdm.blogspot.com/2018/11/minimax-y-alpha-beta-prunning.html (http://iaupsdm.blogspot.com/2018/11/minimax-y-alpha-beta-prunning.html)
- https://www.neverstopbuilding.com/blog/minimax (https://www.neverstopbuilding.com/blog/minimax)
- https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/ (https://www.hackerearth.com/blog/developers/minimax-algorithm-alpha-beta-pruning/)

PAPERS ¶

- http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/Alfa-Beta/Alfa-Beta(2005-II-A).pdf (http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/Inteligencia%20Artificial/Apuntes/tareas_alumnos/Alfa-Beta/Alfa-Beta(2005-II-A).pdf)
- https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjL3Pz4jc7qAhUvn-AKHfVVCawQFjAAegQIBhAB&url=https%3A%2F%2Fe-archivo.uc3m.es%2Fbitstream%2Fhandle%2F10016%2F19841%2FTFG_Nerea_Luis_Mingueza.pdf&usg=A (https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjL3Pz4jc7qAhUvn-AKHfVVCawQFjAAegQIBhAB&url=https%3A%2F%2Fe-archivo.uc3m.es%2Fbitstream%2Fhandle%2F10016%2F19841%2FTFG_Nerea_Luis_Mingueza.pdf&usg=A

JUEGO IMPLEMENTADO CON LA LIBRERIA EASYAI

JUEGO NIM

En este juego se procede a colocar un numero n de fichas, en donde el primer jugador toma un numero de fichas y el otro jugador que en este caso es la maquina hace lo mismo es decir escoger un numero de fichas que desee, y van alternandose en las jugadas.La manera que se gane el juego es que el oponente retire la ultima ficha.

INICIALIZACION DEL JUEGO

Al momento de inicializar el juego se debera de asignar el numero de fichas con las que se desee jugar y se imprimira el numero de fichas que se ingreso, pero en este caso se imprimira ceros que serian las fichas, como por ejemplo:

Ingrese la cantidad de fichas con las que va a jugar: 10

[0 0 0 0 0 0 0 0 0 0]

MOVIMIENTOS EN EL JUEGO

Para los miviemntos que vana realizar cada uno de los jugadores que en este caso seran el humano y el computador en cada iteracion se pedira el numero de fichas que desee quitar cada jugador como por ejemplo:

Player 1 what do you play ? 1

Move #1: player 1 plays 1 :
[0 0 0 0 0 0 0 0 0]

Move #2: player 2 plays 1 :

[0 0 0 0 0 0 0 0]

COMO SE GANA LA PARTIDA

En este juego comose ha mencionado gana el que quite la ultima ficha como por ejemplo: Player 1 what do you play ? 2

Move #9: player 1 plays 2 :
[0]

Juego terminado, gano el jugador: 1

Move #10: player 2 plays 1 :

[]

Juego terminado, gano el jugador: 1

In [9]:

```
from easyAI import TwoPlayersGame, Human_Player, AI_Player, Negamax
import numpy as np

class JuegoFichas(TwoPlayersGame):

    def __init__(self, nplayers):
        self.n = int(input("Ingrese la cantidad de fichas con las que va a jugar: \n"))
        self.fichas = np.zeros(self.n, dtype=int)
        self.nplayer = 1
        self.players = nplayers

    def possible_moves(self):

        moves = []
        contador = 1
        for i in range(len(self.fichas)):
                if self.fichas[i] == 0:
                    moves.append(contador)
                contador += 1
        # print(moves)
        return moves

    def make_move(self, casillero):
        self.fichas=np.delete(self.fichas,casillero-1)
        #print(self.fichas)
    def lose(self):
        if self.fichas.size==0:
            return True
        return False
    def show(self):
        print(self.fichas)

    def scoring(self):
        return -10 if self.lose() else 0

    def is_over(self):

        if (self.fichas.size==0):
            print("Juego terminado, gano el jugador: ", self.nplayer)
            return (self.possible_moves()==0 or self.lose())


if __name__ == "__main__":
    ai_algo = Negamax(6)
    jNim = JuegoFichas([Human_Player(), AI_Player(ai_algo)])
    jNim.play()
```

```
Ingrese la cantidad de fichas con las que va a jugar:
10
[0 0 0 0 0 0 0 0 0 0]

Player 1 what do you play ? 1

Move #1: player 1 plays 1 :
[0 0 0 0 0 0 0 0 0]

Move #2: player 2 plays 1 :
```

```
[0 0 0 0 0 0 0 0]

Player 1 what do you play ? 2

Move #3: player 1 plays 2 :
[0 0 0 0 0 0 0]

Move #4: player 2 plays 1 :
[0 0 0 0 0 0]

Player 1 what do you play ? 3

Move #5: player 1 plays 3 :
[0 0 0 0 0]
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1

Move #6: player 2 plays 1 :
[0 0 0 0]

Player 1 what do you play ? 1

Move #7: player 1 plays 1 :
[0 0 0]
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1
Juego terminado, gano el jugador:  1

Move #8: player 2 plays 1 :
[0 0]

Player 1 what do you play ? 2

Move #9: player 1 plays 2 :
[0]
Juego terminado, gano el jugador:  1

Move #10: player 2 plays 1 :
[]
Juego terminado, gano el jugador:  1
```

In [ ]: