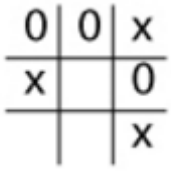
ALGORITMO MINIMAX Y ALPHA BETA PRUNNING

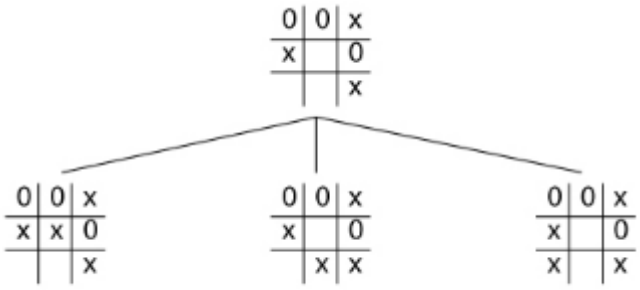
El algoritmo de minimax en simples palabras consiste en la elección del mejor movimiento para el computador, suponiendo que el contrincante escogerá uno que lo pueda perjudicar, para escoger la mejor opción este algoritmo realiza un árbol de búsqueda con todos los posibles movimientos, luego recorre todo el árbol de soluciones del juego a partir de un estado dado, es decir, según las casillas que ya han sido rellenadas. Por tanto, minimax se ejecutará cada vez que le toque mover a la IA.

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

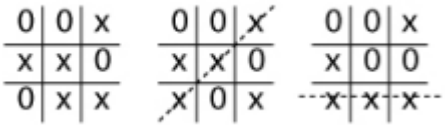
Estado inicial: Es una configuración inicial del juego, es decir, un estado en el que se encuentre el juego. Para nuestro ejemplo sería:



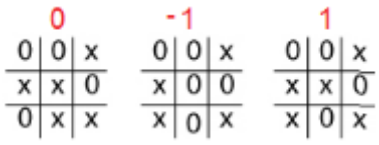
Operadores: Corresponden a las jugadas legales que se pueden hacer en el juego, en el caso del tres en raya no puedes marcar una casilla ya antes marcada.



Condición Terminal: Determina cuando el juego se acabó, en nuestro ejemplo el juego termina cuando un jugador marca tres casillas seguidas iguales, ya se horizontalmente, verticalmente o en diagonal, o se marcan todas las casillas (empate) .

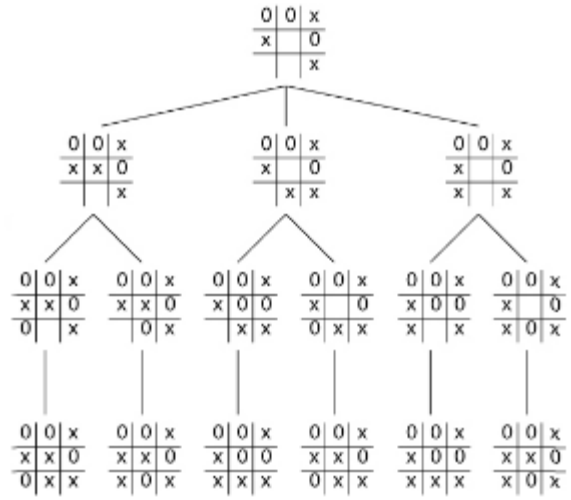


Función de Utilidad: Da un valor numérico a una configuración final de un juego. En un juego en donde se puede ganar, perder o empatar, los valores pueden ser 1, 0, o -1.

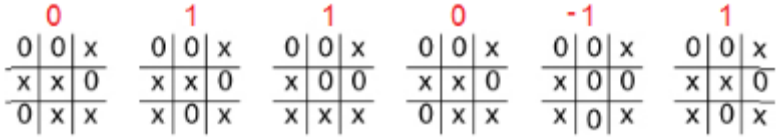


Implementación Minimax: Los pasos que sigue minimax pueden variar, pero lo importante es tener una idea clara de cómo es su funcionamiento, los pasos a seguir son:

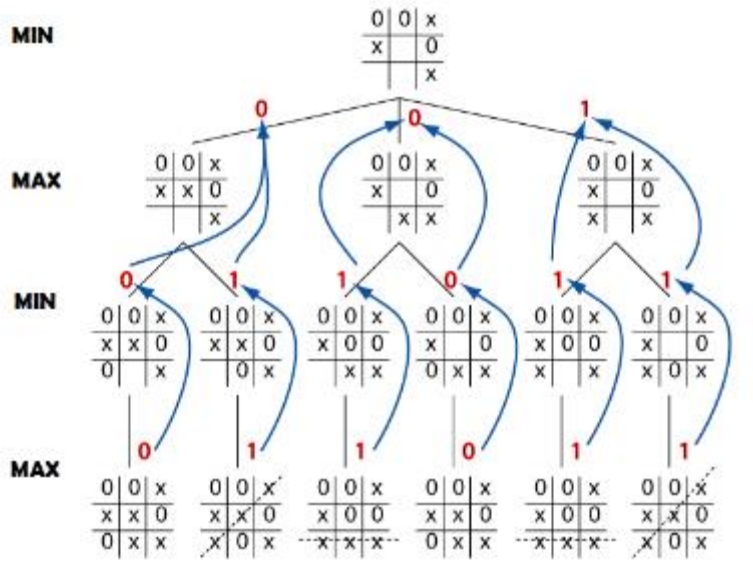
El algoritmo primero generar un árbol de soluciones completo a partir de un nodo dado. veamos el siguiente ejemplo:



Para cada nodo final, buscamos la función de utilidad de estos. En nuestro ejemplo usaremos un 0 para las partidas que terminen en empate, un 1 para las que gane la IA y un -1 para las que gane el jugador humano.



Y lo que hará el algoritmo Minimax cuando vaya regresando hacia atrás, será comunicarle a la llamada recursiva superior cuál es el mejor nodo hoja alcanzado hasta el momento. Cada llamada recursiva tiene que saber a quién le toca jugar, para analizar si el movimiento realizado pertenece a la IA o al otro jugador, ya que cuando sea el turno de la IA nos interesa MAXIMIZAR el resultado, y cuando sea el turno del rival MINIMIZAR su resultado.



alfa-beta

es un algoritmo de búsqueda que busca disminuir el número de nodos que son evaluados por el algoritmo minimax en su árbol de búsqueda . Se trata de un algoritmo de búsqueda de confrontación de uso común para la máquina de juego de los juegos de dos jugadores (Tic-tac-dedo del pie , ajedrez , Go , etc.). Se deja la evaluación de un movimiento cuando al menos una de las posibilidades ha encontrado que demuestra el paso a ser peor que un movimiento examinado con anterioridad. Estas medidas no tienen que ser evaluados sucesivamente. Cuando se aplica a un árbol minimax estándar, devuelve el mismo movimiento como Minimax haría, pero las ciruelas pasas de distancia ramas que no es posible influir en la decisión final.

EJEMPLO TRES EN RAYA

```
import pygame, time
#colores
blanco = (255,255,255)
rojo = (200,0,0)
rojo_tomate = (255,51,0)
verde = (0,204,0)
verde_lima = (0,255,0)
azul = (0,51,255)
azul_marino = (0,0,119)
gris = (150,150,150)
gris_claro = (204,204,204)

tema = 0 #numero 0-4
color_fondo1 = [blanco, rojo, verde, azul, gris][tema]
color_fondo2 = [gris_claro, rojo_tomate, verde_lima, azul_marino, gris_claro][tema]

#variables ventana
t_casilla = 200
t_letra = 128
resolucion = (t_casilla * 3, t_casilla * 3)
FPS = 30

pygame.init()
clock = pygame.time.Clock()

#imagenes
img = {"X" : pygame.transform.smoothscale(pygame.image.load("x.png"), (t_letra, t_letra)),
     "O" : pygame.transform.smoothscale(pygame.image.load("o.png"), (t_letra, t_letra))}

#variables juego
VACIO = " "
JUGADOR = "X"
MAQUINA = "O"
dificultad = 5
contador = ""
filas_ganadoras = ((0, 1, 2), (3, 4, 5), (6, 7, 8),
                   (0, 3, 6), (1, 4, 7), (2, 5, 8),
                   (0, 4, 8), (2, 4, 6))

#funciones juego
colisionan = lambda punto, pos, dimensiones: pos[0] <= punto[0] <= pos[0] + dimensiones[0]
and pos[1] <= punto[1] <= pos[1] + dimensiones[1]
```

```
def render_tablero(screen, tablero):
    """Renderiza el tablero y pinta el fondo con los colores del tema
    ARGUMENTOS:
        -screen. Superficie principal de pygame sobre la que se pinta todo (display).
        -tablero. String de longitud 9 que contiene los valores del tablero."""
    #pintar fondo
    a = False
    for i in range(0, t_casilla * 3, t_casilla):
        for j in range(0, t_casilla * 3, t_casilla):
            a = a == False
            if a: pygame.draw.rect(screen, color_fondo1, ((i, j), (t_casilla, t_casilla)))
            else: pygame.draw.rect(screen, color_fondo2, ((i, j), (t_casilla, t_casilla)))
    #poner Os y Xs
    for a in range(9):
        if tablero[a] == " ": continue
        screen.blit(img[tablero[a]], (a % 3 * t_casilla + (t_casilla - t_letra) / 2, a // 3 * t_casilla +
(t_casilla - t_letra) / 2))

def minimax(tablero, turno_player, profundidad = 5):
    """Implementacion del algoritmo minimax a nuestro tres en raya.
    ARGUMENTOS:
        -tablero. String de longitud 9 que contiene los valores del tablero.
        -turno_player. Booleano que indica el turno, si es positivo significa que le toca al jugador
humano.
        -profundidad. Valor numerico que limita el numero de veces que la funcion se llama a si
misma (dificultad) y que incita a la maquina
                a realizar los movimientos que impliquen alargar la partida lo maximo posible
(intentando ganar siempre)."""
    if ganador(tablero) == MAQUINA: return (+10 - profundidad, None) #gana pc
    elif ganador(tablero) == JUGADOR: return (-10 - profundidad, None) #pierde pc
    elif VACIO not in tablero or profundidad < 1: return (0, None) #empatan
    elif turno_player: #turno de jugador
        best = (+11, None)
        for a in range(9):
            if tablero[a] == " ":
                valor = minimax(tablero[:a] + JUGADOR + tablero[a + 1:], not turno_player,
profundidad - 1)[0]
                if valor < best[0]: best = (valor, a) #jugador intenta causar el MENOR beneficio a pc
        return best
    else: #turno de pc
        best = (-11, None)
        for a in range(9):
            if tablero[a] == " ":
                valor = minimax(tablero[:a] + MAQUINA + tablero[a + 1:], not turno_player,
profundidad - 1)[0]
                if valor > best[0]: best = (valor, a) #pc intenta causar el MAYOR beneficio a si mismo
        return best
```

```
def ganador(tablero):
    """Indica si alguien ha ganado la partida y en caso verdadero devuelve la letra del ganador.
    Como argumento toma unicamente el tablero como string de longitud 9."""
    for fila in filas_ganadoras:
        if tablero[fila[0]] == VACIO: continue
        if len(set(tablero[casilla] for casilla in fila)) == 1: return tablero[fila[0]]
    return False

def movimiento_pc(tablero):
    """Realiza el movimiento del pc en el tablero"""
    pygame.mouse.set_cursor(*pygame.cursors.broken_x)
    t0 = time.time() #inicio cronometro
    if tablero[4] == VACIO: a = 4 #cuando el centro esta vacio siempre trata de ocuparlo
    elif dificultad < 1:
        a = set(i for i in range(9) if tablero[i] == VACIO).pop() #uno aleatorio entre los movimientos
validos
    elif dificultad >= 1:
        a = minimax(tablero, False, dificultad)[1] #algoritmo minimax limitado segun la dificultad

    if a is not None: tablero = tablero[:a] + MAQUINA + tablero[a + 1:] #sustituye la posicion a en
el tablero por su letra
    print( "La maquina ha tardado {:.5f} ms".format((time.time() - t0) * 1000)) #tiempo desde
inicio del cronometro
    pygame.mouse.set_cursor(*pygame.cursors.arrow)
    return tablero

screen = pygame.display.set_mode(resolucion)
pygame.display.set_caption("3 en raya")

def main():
    global contador
    #variables que se han de resetear al iniciar la partida
    turno_player = True
    tablero = VACIO * 9
    salir = False
    while not salir:
        clock.tick(FPS) #limitar los FPS para no consumir recursos innecesarios

        if not turno_player and not ganador(tablero) and VACIO in tablero:
            tablero = movimiento_pc(tablero)
            turno_player = True

        for event in pygame.event.get():
            if event.type == pygame.QUIT: #click en la cruz roja
                raise SystemExit

            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
```

```
                raise SystemExit

            elif event.type == pygame.MOUSEBUTTONDOWN:
                if event.button == 1:
                    if ganador(tablero) or not VACIO in tablero: salir = True #Cuando la partida termina, se espera a clickar para reiniciarla
                    x = event.pos[0] // t_casilla + 3 * (event.pos[1] // t_casilla)
                    if tablero[x] == " ":
                        tablero = tablero[:x] + JUGADOR + tablero[x + 1:]
                        turno_player = False

        render_tablero(screen, tablero)
        pygame.display.flip()

    if ganador(tablero):
        contador = contador + ganador(tablero)
        print("\n" * 100)
        print(ganador(tablero), "ha ganado!    TOTAL: ", "X", str(contador.count("X")) + "-" + str(contador.count("O")), "O")
        print("\n")
        salir = True
    elif VACIO not in tablero:
        print("\n" * 100)
        print("EMPATE!!    TOTAL: ", "X", str(contador.count("X")) + "-" + str(contador.count("O")), "O")
        print("\n")
        salir = True

while True: main()
raise SystemExit
```