



UA

Sistemas Inteligentes

Grupo 5 prácticas

Vadym Formanyuk: x5561410x

29 de octubre de 2022

Índice

1. Función de evaluación	2
1.1. Casos base. Presentación de los mismos y explicación de porqué se han escogido	2
1.2. ¿Se ha implementado una nueva clase para tratar con el tablero?¿Porqué?	2
1.3. Para cada función de evaluación diseñada	2
2. Minimax	13
2.1. ¿Cómo se ha implementado un nodo?	13
2.2. ¿Qué devuelve tu implementación de minimax? ¿Cómo accedes a la última jugada?	14
2.3. ¿Hasta qué nivel puedes bajar en tu implementación de minimax? Asume la raíz del árbol empieza en 0 y el nivel aumenta en 1 por cada nivel de descendientes.	14
2.4. ¿Cómo juega la máquina en ese nivel? Pon ejemplos	15
2.5. ¿Minimax siempre hace la misma jugada para un determinado tablero? ¿Cómo podríamos evitar esto?	15
2.6. [OPCIONAL] Implementa que minimax no realice siempre la misma jugada para nodos con $F(N)$ iguales	16
3. AlfaBeta	16
3.1. ¿Dónde se introduce la poda alfa-beta en tu algoritmo minimax?	16
3.2. ¿Hasta qué nivel de profundidad juega tu algoritmo de manera razonable?	18
3.3. ¿Cuántos nodos te ahorras al aplicar alfa-beta vs minimax?, ¿Es más rápido que minimax? Muestra ejemplos	18
4. Comparación de funciones de evaluación	19

1. Función de evaluación

1.1. Casos base. Presentación de los mismos y explicación de porqué se han escogido

1.2. ¿Se ha implementado una nueva clase para tratar con el tablero?¿Porqué?

1.3. Para cada función de evaluación diseñada

- Ejecución de los casos base y explicación de la función de evaluación.
- Análisis de casos. Se debe valorar el funcionamiento de $F(N)$ para varios casos seleccionados. Prueba con modificaciones leves de los casos para verificar su funcionamiento de manera más exhaustiva
- Análisis de coste temporal

En cuanto a todas las preguntas anteriores, se irán exponiendo las respuestas a continuación:

Para tratar con el tablero no he creado una nueva clase pero si que he creado un nuevo archivo ".py" donde he colocado los métodos auxiliares que uso para el algoritmo. Esto métodos auxiliares los llamo en el archivo algoritmo.py. Estos métodos auxiliares hacen la función como de una fila o columna completa de fichas, o miran si el tablero (nodo) es terminal.

```
PIEZA_JUGADOR = 1
PIEZA_AI = 2

#obtiene una fila completa a partir de num_fila
def getFilaCompleta(tablero,num_fila):
    fila_completa = []
    if num_fila < tablero.getAlto():
        for c in range(tablero.getAncho()):
            fila_completa.append(tablero.getCelda(num_fila,c))

    return fila_completa

#obtiene una columna completa a partir de num_col
def getColumnaCompleta(tablero,num_col):
    col_completa = []
    if num_col < tablero.getAncho():
        for r in range(tablero.getAlto()):
            col_completa.append(tablero.getCelda(r,num_col))

    return col_completa

#Obtengo todas las columnas sin llenar
def obtenerColumnasSinLlenar(tab):
    playable_locations = []
    for col in range(tab.getAncho()): #recorro todas las columnas
        if columnaSinLlenar(tab,col):
            playable_locations.append(col)
    return playable_locations

def columnaSinLlenar(tab,col):
    return tab.getCelda(0, col) == 0

#saca la primera posicion de la columna, más arriba que sea un 0
def getPrimerCeroEnLaColumna(tab,col): #unpoco diferente al del video
    row = tab.getAlto() - 1
    for r in range(tab.getAlto()):
        if tab.getCelda(r,col) != 0:
            row = r-1
            break #sin el break sigue iterando y no da el resultado correcto
    return row

#devuelve TRUE si el siguiente nodo es el nodo ganador o ya no se puede hacer ningun movimiento
def esNodoTerminal(tablero):
    return tablero.cuatroEnRaya() == PIEZA_AI or tablero.cuatroEnRaya() == PIEZA_JUGADOR or len(obtenerColumnasSinLlenar(tablero)) == 0

# busca en col la primera celda vacía
def buscaPrimeraVacía(tablero, col):
    if tablero.getCelda(0,col) != 0:
        i=-1
        i=0
        while i<tablero.getAlto() and tablero.getCelda(i,col)!=0:
            i=i+1
        i=i-1

    return i

return go(f, seed, [])
}
```

Figura 1: Métodos auxiliares

La función de evaluación:

```
def verPuntuacion_de_este_tablero(tablero,pieza):
    puntuacion = 0

    #PREFERENCIA POR EL CENTRO
    columna_centro = tablero.getAncho()//2
    filas_de_la_columna_del_centro = getColumnaCompleta(tablero,columna_centro)
    piezas_centro = filas_de_la_columna_del_centro.count(pieza)
    puntuacion += piezas_centro * 6

    #HORIZONTAL
    for r in range(tablero.getAlto()):
        fila_array = getFilaCompleta(tablero,r)
        for c in range(tablero.getAncho() - 3):
            seleccion = fila_array[c:c+4]
            puntuacion += evaluar_seleccion(seleccion,pieza)

    #VERTICAL
    for c in range(tablero.getAncho()):
        col_array = getColumnaCompleta(tablero,c)
        for r in range(tablero.getAlto() - 3):
            seleccion = col_array[r:r+4]
            puntuacion += evaluar_seleccion(seleccion,pieza)

    #DIAGONAL POSITIVO
    for r in range(tablero.getAlto() - 3):
        for c in range(tablero.getAncho() - 3):
            seleccion = [tablero.getCelda(r+i,c+i) for i in range(4)]
            puntuacion += evaluar_seleccion(seleccion,pieza)

    #DIAGONAL NEGATIVO
    for r in range(tablero.getAlto() - 3):
        for c in range(tablero.getAncho() - 3):
            seleccion = [tablero.getCelda(r+3-i,c+i) for i in range(4)]
            puntuacion += evaluar_seleccion(seleccion,pieza)

    return puntuacion

def evaluar_seleccion(seleccion,pieza):

    puntuacion = 0
    if seleccion.count(pieza) == 4:
        puntuacion += 100
    elif seleccion.count(pieza) == 3 and seleccion.count(0) == 1:
        puntuacion += 10
    elif seleccion.count(pieza) == 2 and seleccion.count(0) == 2:
        puntuacion += 5

    pieza_del_oponente = PIEZA_JUGADOR
    if pieza == PIEZA_JUGADOR:
        pieza_del_oponente = PIEZA_AI
    if seleccion.count(pieza_del_oponente) == 3 and seleccion.count(0) == 1:
        puntuacion -= 75

    return puntuacion
```

Figura 2: Función de evaluación

La función de evaluación consta de 4 casos posibles para ganar , donde cada caso evalúa distintas selecciones de fichas. Es necesario tener 4 casos de selección porque según las reglas del "4 en ralla" es posible ganar al obtener 4 fichas seguidas del mismo tipo en 4 posiciones distintas: horizontal, vertical y en diagonal (en caso del algoritmo, es necesario comprobar ambos diagonales, el "positivo" y el "negativo"). Para recorrer el tablero hará falta un bucle "for" anidado dentro de otro bucle "for" que es un array bidimensional el que crea al tablero (alto y ancho). Para cada combinación de 4 casillas se evalúa (en el método llamado "evaluar-seleccion" de la figura 1) la cantidad de fichas del mismo tipo que hay en esa selección y se le da una

puntuación acorde a ello. Al finalizar el bucle exterior se obtiene una puntuación u otra dependiendo de los criterios establecidos.

En cuanto a la cantidad de puntos que se dan, la idea final es obtener la columna con mayor puntuación (que es en la cual irá a parar la ficha), para ello he establecido diferentes puntajes:

- Para 4 piezas del mismo valor: 100 puntos
- Para 3 piezas del mismo valor: 10 puntos
- Para 2 piezas del mismo valor: 5 puntos

He implementado en el algoritmo que quite 75 puntos cuando vea que el jugador enemigo tiene 3 piezas juntas en ese tablero, y así poder contrarestarle luego adecuadamente.

A parte de todo lo anterior, también he implementado que el algoritmo le dé prioridad al centro del tablero (ya que estadísticamente hay más probabilidad de ganar si hay fichas en el centro)

- Las distintas evaluaciones:

(1) Evaluación Horizontal:

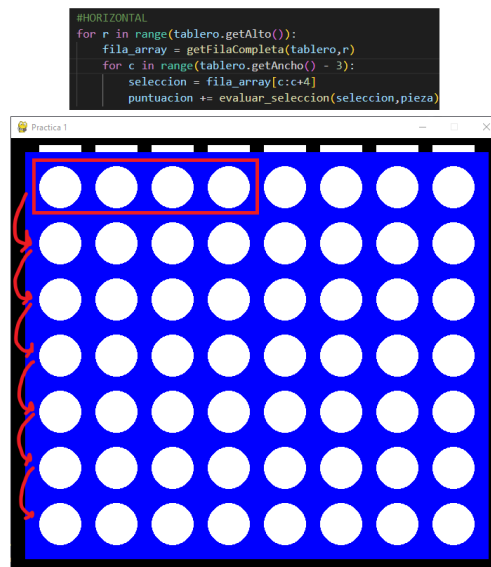


Figura 3: Selección horizontal

Este bucle recorre todas las filas del tablero buscando combinaciones de 4 casillas en forma horizontal.

(2) Evaluación Vertical:

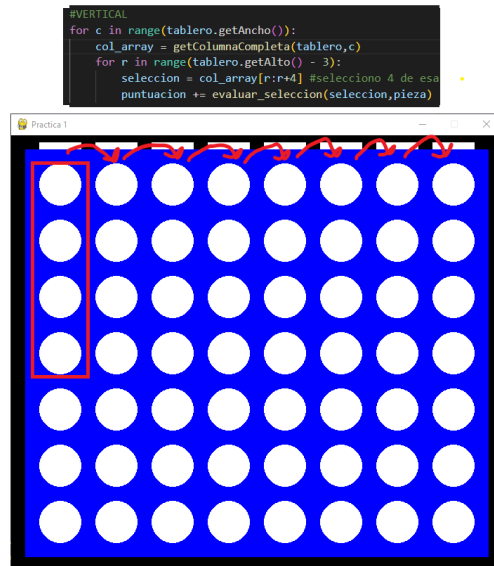


Figura 4: Selección vertical

Este bucle recorre todas las columnas del tablero buscando combinaciones de 4 casillas de forma vertical.

(3) Evaluación Diagonal positivo:

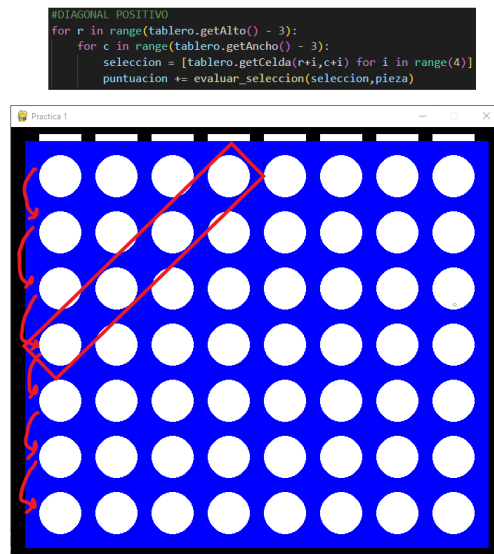


Figura 5: Selección diagonal pos.

Este bucle recorre todas las filas del tablero buscando combinaciones de 4 casillas de forma diagonal positiva (hacia arriba).

(4) Evaluación Diagonal Negativo:

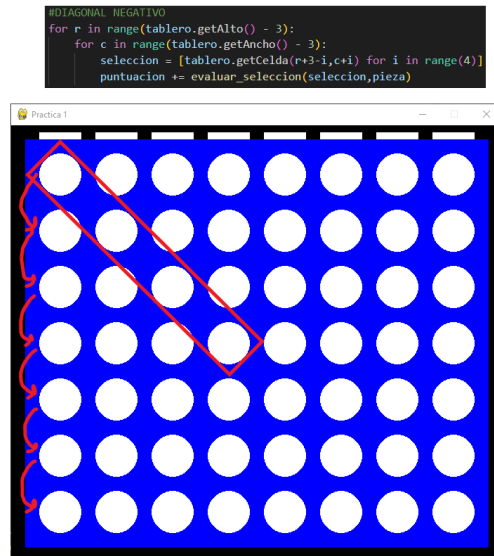


Figura 6: Selección diagonal neg.

Este bucle recorre todas las filas del tablero buscando combinaciones de 4 casillas de forma diagonal negativa (hacia abajo).

Una vez explicada la función de evaluación, paso a tratar los casos base:

- Caso base: La preferencia por el centro. Se ha escogido este caso base para desmotrar la parte de preferencia por el centro.

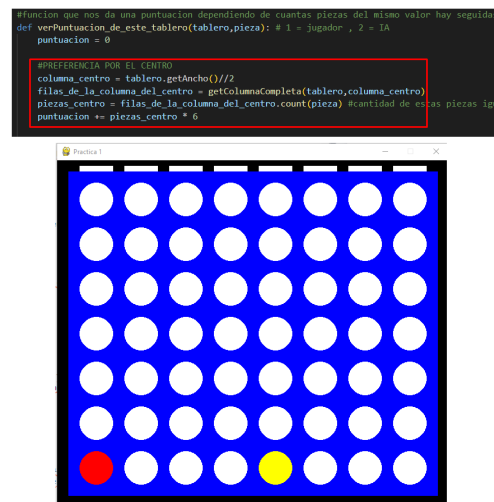


Figura 7: Preferencia por el centro

Dado que según las mejores estrategias del 4 en ralla (y por lógica común teniendo en cuenta las reglas de este juego), empezar por el centro aumenta las probabilidades de ganar, esto es así porque tener fichas en el centro nos permite poder tener más opciones a la hora de a qué dirección queremos colocar las 4 fichas (sea izquierda o derecha). Por tanto, el correcto funcionamiento de ese caso base se puede ver directamente al empezar la partida, donde yo coloco una ficha y el algoritmo coloca la suya directamente al centro.

Nota: como el número de columnas es par, no hay un centro como tal por lo que escojo la columna 5 (ancho/2). Una modificación para tener un juego más variado es que en vez de que sea la columna 5, escoger la columna 4.

Aquí un ejemplo detallado (usado en el método minimax, que se explicará más adelante pero que es necesario citar para explicar el algoritmo de evaluación). Como se puede ver en este ejemplo de profundidad 1, a partir del tablero raíz creo los auxiliares (donde coloco 1 ficha en cada una de sus columnas) y veo que sólo le da la puntuación al que está en la columna 5.

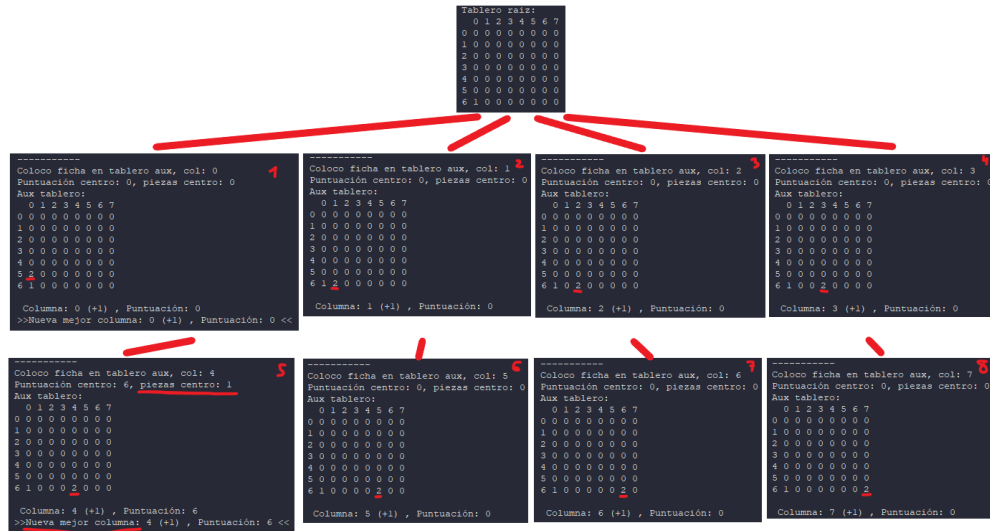


Figura 8: Preferencia por el centro demostración

- Caso base: Explicación visual de la evaluación. Se ha escogido este caso base para mostrar visualmente de cómo calcula el algoritmo de evaluación la puntuación del tablero (usado en la función minimax con profundidad 1).

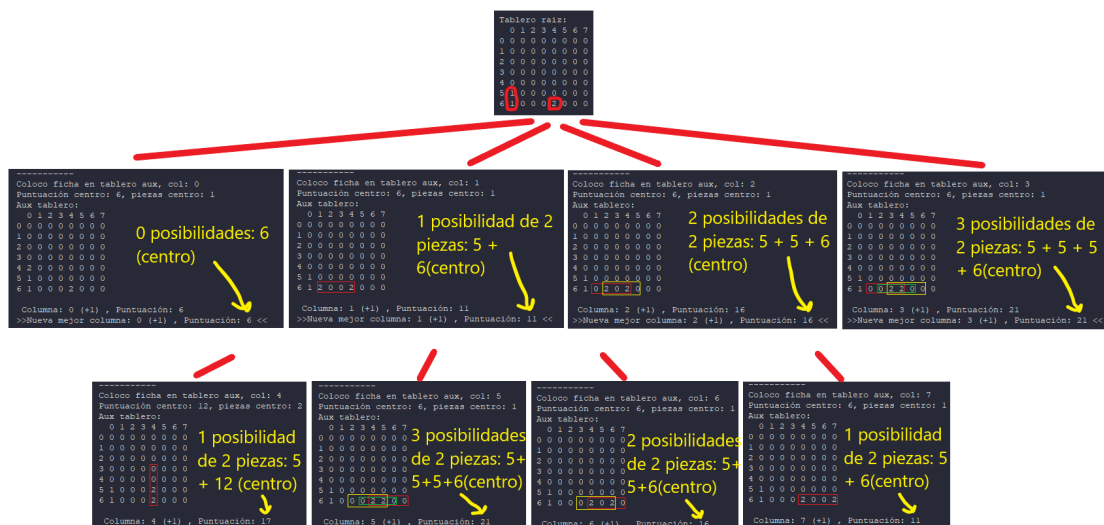


Figura 9: Función de evaluación explicada visualmente

Tal como se explicado anteriormente la evaluación, se puede observar en la figura 9 los efectos de esa evaluación a la hora de elegir columna por parte del algoritmo. A pesar de que en la figura 9 solo se observa la evaluación de forma horizontal y vertical, el algoritmo evalúa del mismo modo a los casos diagonales.

- Caso base: La importancia de la profundidad. Se ha escogido este caso base para mostrar de forma sencilla (es decir, sin muchas piezas de por medio pero que sea claro) de porqué una profundidad mayor implica una elección de columna más correcta.

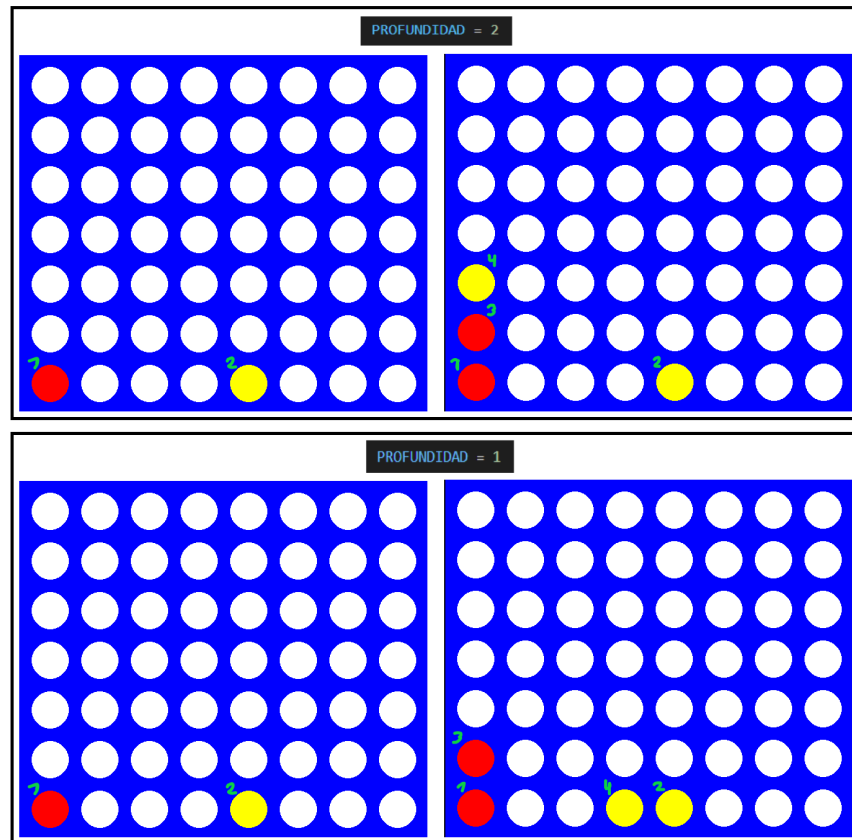


Figura 10: Importancia de la profundidad

En la figura 10, se puede observar 2 jugadas iguales pero con colocación de la ficha por parte de la IA distinta debido a que ambos tienen profundidades distintas. En la jugada de la profundidad 1 (la de abajo) podemos observar que la IA ha colocado su ficha en una posición que le es más ventajosa (tal como indica la evaluación) sin tener en cuenta que el jugador (su oponente) tiene ya 2 fichas juntas. Mientras que en la jugada de arriba que tiene una profundidad 2, el algoritmo ve el peligro de que el jugador tenga 2 fichas juntas (ya que en la simulación con los tableros auxiliares usando el minimax)

- Caso base: Explicación del MiniMax. Se ha escogido este caso base porque es un ejemplo visual de cómo va haciendo minimax el algoritmo presentado, además , responde también de una manera bastante visual de cómo funciona la función de evaluación según el tablero que se le ponga.

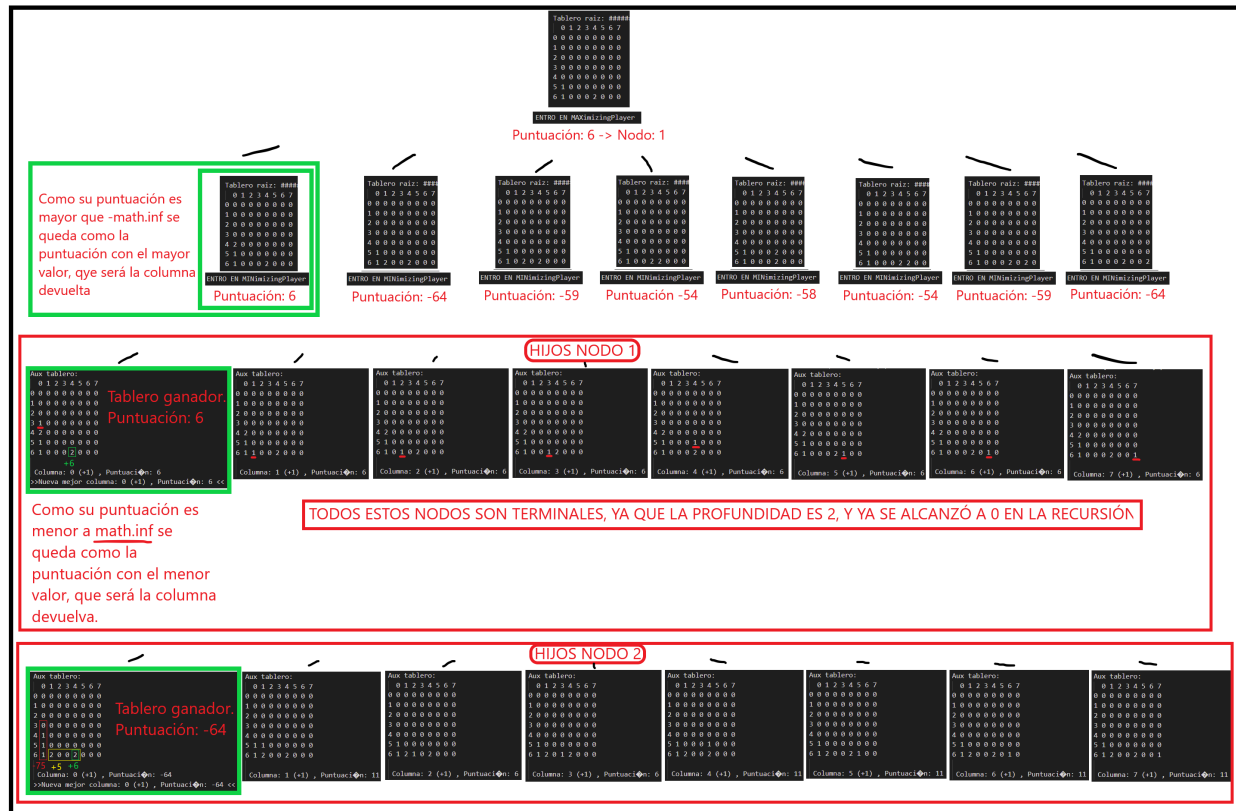


Figura 11: MiniMax visual

En la figura 11, se puede observar una jugada en la que el algoritmo tiene que detectar (en profundidad 2) el peligro de que el jugador (su oponente) tenga 2 piezas juntas. En la imagen se observa que un árbol de 3 niveles, donde el último nivel es terminal y está en la iteración MIN, de este nivel 3 se escoge el nodo con el valor mas menor de todos , que resulta ser el 64". Luego, nos fijamos en el nivel 2 , donde está en la iteración MAX y donde de todos los nodos se escoge el nodo con el mayor valor de todos, que resulta ser el "6". Al final este nodo ganador es donde la IA moverá su ficha. A pesar de ser el ejemplo de profundidad 2, el algoritmo minimax funciona de igual manera con profundidades mayor a 2 (MAX,MIN,MAX,MIN,etc).

Análisis de casos. Aquí se expone cómo obtiene su puntuación la función de evaluación:

(1) Tablero 1:

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0
5	1	0	0	0	0	0	0	0
6	1	0	2	0	2	0	0	0
	-75	+5	+6	+5				

Figura 12: Tablero 1

En este tablero (de nivel 3, MIN) dentro de un juego de profundidad 2 y con puntuación final '-59' se observa los siguientes puntajes: Los '-75' puntos de las 3 fichas juntas (+ 1 vacía) del contrario; Los 2 '+5' de recorrer el bucle horizontal y haber detectado que se pueden formar las 4 fichas de 2 maneras posible (+ las 2 vacías); El '+6' de tener una ficha en el medio (NOTA: después de la primera jugada de la IA, todos los tableros tendrán ese +6, por lo que los puntos empiezan por +6 en vez de 0).

(2) Tablero 2:

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0
3	2	0	0	0	0	0	0	0
4	2	0	1	1	0	0	0	0
5	1	1	1	2	2	0	0	+5
6	1	1	2	1	2	2	0	+5
	-75	-75	+5	+6	+5			

Figura 13: Tablero 2

En este tablero (de nivel 3, MIN) dentro de un juego de profundidad 2 y con puntuación final '-118' se observa los siguientes puntajes: Los 2 '-75' puntos de las 3 fichas juntas del contrario (+ 1 vacía); Los 4 '+5' puntos de tener 2 fichas juntas (+ 2 vacías) en 3 orientaciones diferentes (horizontal, vertical y diagonal positivo); Los 2 '+6' de tener 2 fichas en la columna del medio.

(3) Tablero 3:

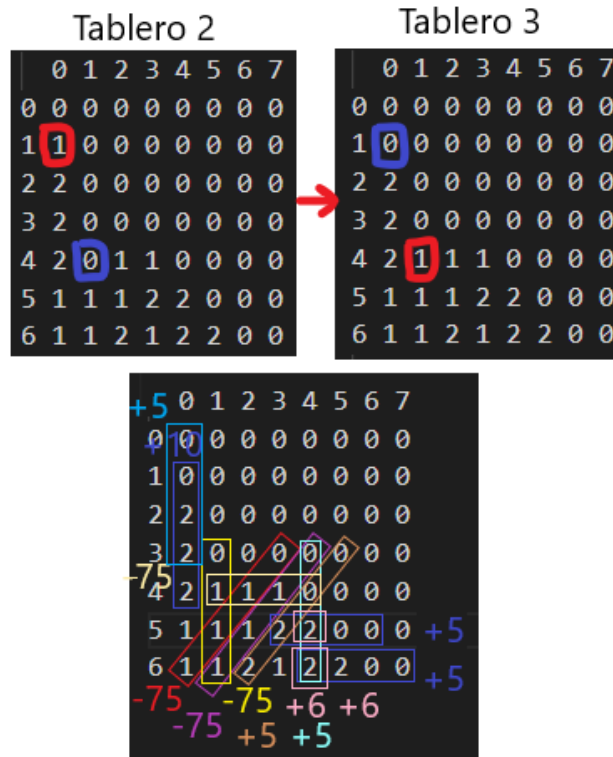


Figura 14: Tablero 3

En este tablero que es el mismo que el tablero presentado anteriormente se puede ver la gran diferencia de puntuación que se obtiene con una leve modificación al haber puesto la ficha en la columna 2 en vez de la 1.

Coste temporal. Presentación del coste temporal de la función de evaluación teniendo en cuenta la profundidad y dependiendo de las fichas que hay colocadas en el tablero:

- Profundidad 1: En esta profundidad el tiempo de evaluar al tablero no pasaba de los 0,001 segundos en las jugadas donde más piezas había que evaluar y un valor muy cercano al 0 en el resto de jugadas, por tanto no ofrece ninguna información visual de relevancia que pueda ser puesta en un gráfico.
- Profundidad 2:

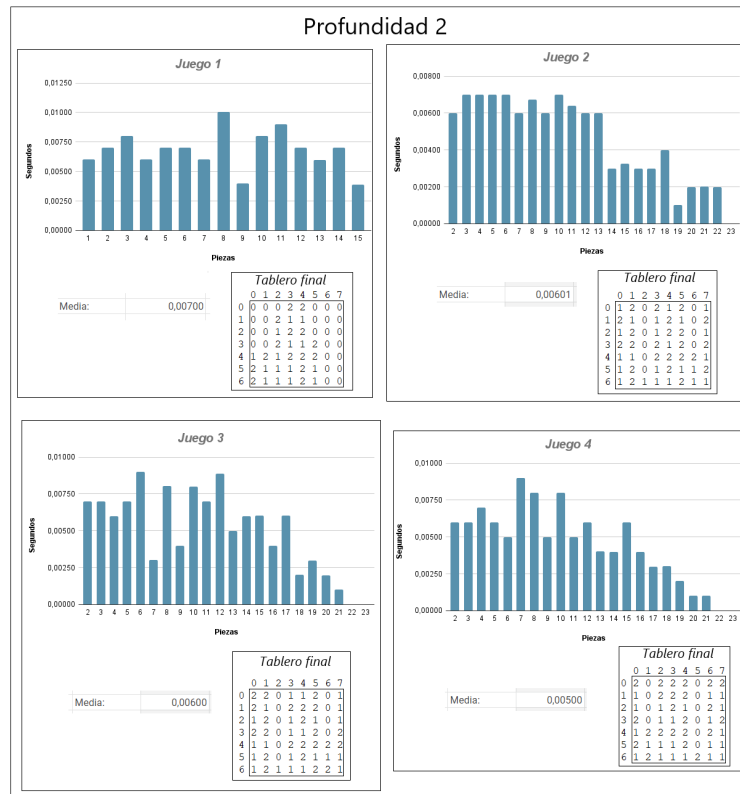


Figura 15: Profundidad 2

Se puede observar que la función tarda de media unos 0,0065 segundos para evaluar cada jugada, y reduciéndose ese tiempo una vez que se vayan reduciendo las casillas vacías donde poner la ficha, llegando hasta 0 segundos cuando ya no quede casi sitio donde poner ficha.

■ Profundidad 3:

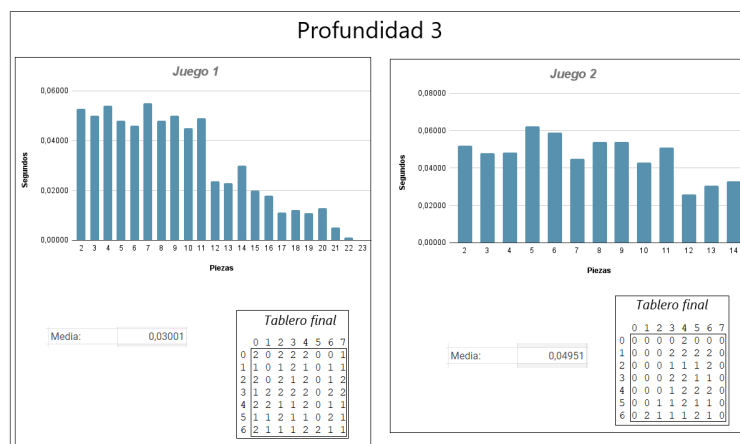


Figura 16: Profundidad 3

Se contempla que la función tarda de media unos 0,04 segundos para evaluar cada jugada, y reduciéndose ese tiempo una vez que se vayan reduciendo las casillas vacías donde poner la ficha, llegando hasta 0 segundos cuando ya no quede casi sitio donde poner ficha.

Conclusiones coste temporal:

Viendo las 3 primeras profundidades se llega a la conclusión siguiente: En la profundidad 1 tenemos que de media el coste para evaluar el tablero (sus 8 columnas) es de 0,001 segundos; En la profundidad 2 es de 0,007 segundos, y en la profundidad 3 es de 0,05 segundos. Todo esto nos indica de que tal como por lógica se ve es que cada vez que aumentemos de profundidad (donde hay que evaluar 8 tableros más por hijo) aumenta el coste multiplicando por 8 el coste de la profundidad anterior.

2. Minimax

2.1. ¿Cómo se ha implementado un nodo?

Se ha implementando un nodo de la siguiente manera: Una vez que se llame al algoritmo minimax, se le pasa un tablero que será el base. Luego, se recorren todas las columnas del tablero y creando una copia de ese tablero base y añadiéndole una ficha (la del algoritmo) en esa columna. Si la profundidad es 1 entonces se escogerá la columna con la puntuación máxima de esos tableros auxiliares. En el caso de que la profundidad fuese mayor a 1 entraría a un nivel más bajo escogiendo los valores mayores-menores dependiendo de donde esté en la recursión (está más detallado en la explicación oficial del minimax). Para poder ver un ejemplo visual de cómo hace mi algoritmo el minimax con profundidad mayor a 1 hay que fijarnos en el caso baso explicado en el apartado anterior (Figura 11).

y se vuelve a llamar a la función minimax con esa copia. Todo esto se va haciendo hasta que la profundidad sea 0 o ese tablero(nodo) que se le pasa por parametros al minimax sea terminal, si es terminal entonces se elige al final esa columna con la máxima puntuación (100000 de puntos, para que sea la elegida si o si), si no es terminal se calcula la puntuación de ese tablero en el otro método (descrito en apartados anteriores) y si esa puntuación es mayor que la de las otras columnas entonces se asigna esa columna como la mejor (que al final será la elegida donde colocar la ficha).

```

def miniMax(tablero, profundidad, maximizingPlayer):
    columnasSinLlenar = obtenerColumnasSinLlenar(tablero)
    esTerminal = esNodoTerminal(tablero)

    if profundidad == 0 or esTerminal:
        if esTerminal:
            if tablero.cuatroEnRaya() == PIEZA_AI:
                return (None, None, 1000000)
            elif tablero.cuatroEnRaya() == PIEZA_JUGADOR:
                return (None, None, -1000000)
            else:
                return (None, None, 0)
        else: #profundidad es 0
            punt = verPuntuacion_de_este_tablero(tablero, PIEZA_AI)
            return (None, None, punt)

    if maximizingPlayer:
        value = -math.inf
        mejorColumna = random.choice(columnasSinLlenar)
        mejorFila = getPrimerCeroEnLaColumna(tablero, mejorColumna)

        for col in columnasSinLlenar:
            fila = getPrimerCeroEnLaColumna(tablero, col)
            aux_tablero = Tablero(tablero)
            aux_tablero.setCelda(fila, col, PIEZA_AI)
            nueva_puntuacion = miniMax(aux_tablero, profundidad-1, False)[2]

            if nueva_puntuacion > value:
                value = nueva_puntuacion
                mejorColumna = col
                mejorFila = fila

        return mejorColumna, mejorFila, value
    else:
        value = math.inf
        mejorColumna = random.choice(columnasSinLlenar)
        mejorFila = getPrimerCeroEnLaColumna(tablero, mejorColumna)

        for col in columnasSinLlenar:
            fila = getPrimerCeroEnLaColumna(tablero, col)
            aux_tablero = Tablero(tablero)
            aux_tablero.setCelda(fila, col, PIEZA_JUGADOR)
            nueva_puntuacion = miniMax(aux_tablero, profundidad-1, True)[2]

            if nueva_puntuacion < value:
                value = nueva_puntuacion
                mejorColumna = col
                mejorFila = fila

        return mejorColumna, mejorFila, value

```

Figura 17: Algoritmo minimax

2.2. ¿Qué devuelve tu implementación de minimax? ¿Cómo accedes a la última jugada?

Mi implementación del minimax devuelve un array que contiene la columna y la fila con mayor puntuación (que es en la cual luego colocará ficha la IA con `setCelda()`) Otro de los parametros (para uso interno del algoritmo minimax) que devuelve dentro del array es la puntuación que va calculando en las iteraciones de la recursión del minimax. Esa puntuación o la obtengo del método de evaluar o la obtengo de forma manual poniendo un millón de puntos para cuando hay 4 piezas seguidas y está por declararse victoria en ese tablero.

2.3. ¿Hasta qué nivel puedes bajar en tu implementación de minimax? Asume la raíz del árbol empieza en 0 y el nivel aumenta en 1 por cada nivel de descendientes.

En mi implementación de minimax puedo bajar hasta cualquier nivel que le ponga en los parámetros a la hora de llamar la función minimax desde el "main.py" (en la variable global declarada en el main 'PROFUNDIDAD'), pero a partir del nivel 3 ya se empieza a notar que tarda en procesar todos los nodos ya que empieza a ir más y más lento el programa a medida que le añadimos profundidad.

2.4. ¿Cómo juega la máquina en ese nivel? Pon ejemplos

Hablando del nivel de dificultad teniendo en cuenta la profundidad 3 (que es la profundidad en la cual el algoritmo aún no tarda 'mucho' tiempo en procesarse) la máquina tiene una dificultad difícil en la cual me es imposible ganarle en todas las veces que he jugado.

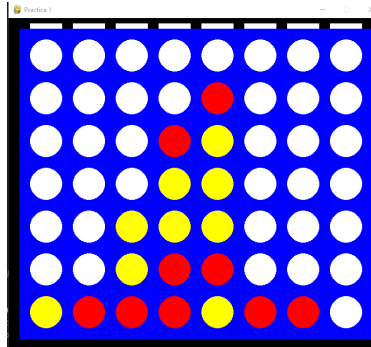


Figura 18: Ejemplo

En la figura 18 de ejemplo vemos como el algoritmo me bloquea las 3 fichas que tenía de la fila más abajo, y otra cosa que también intenta en todas las partidas que he jugado es tener 3 fichas en medio para que en un futuro pueda ganar poniendo ficha o por la izquierda o por la derecha (lo del medio está programado así, ya que según los mejores jugadores de 4 en ralla es la mejor forma de ganar este juego)

2.5. ¿Minimax siempre hace la misma jugada para un determinado tablero? ¿Cómo podriamos evitar esto?

Si, esto es debido a que al aplicar minimax para un determinado tablero siempre habrá una columna que obtiene más puntos que las demás, y si hay 2 (o más) columnas con la misma puntuación tengo puesto que la columna elegida sea la que tenga más puntuación que las demás (sin incluir las columnas con la misma puntuación que esa), lo que hace que solo sea la primera de estas columnas. Es decir, que habiendo varias columnas con la misma puntuación, solo se elegirá la primera de ellas al entrar al `if` porque tengo puesto un `»` para entrar al `if`.

Para evitar esto, creo una lista donde se guarden las columnas que tienen la misma puntuación que la columna con la mayor puntuación, y elegir una random de esa lista.

2.6. [OPCIONAL] Implementa que minimax no realice siempre la misma jugada para nodos con $F(N)$ iguales

```
for col in columnasSinLlenar:
    fila = getPrimerCeroEnLaColumna(tablero,col)
    aux_tablero = Tablero(tablero)
    aux_tablero.setCelda(fila,col,PIEZA_JUGADOR)
    nueva_puntuacion = miniMax(aux_tablero,profundidad-1,True)[2]

    if nueva_puntuacion < value:
        value = nueva_puntuacion
        mejorColumna = col
        mejorFila = fila
        if globals.MEMORIA_DIFERENTES:
            globals.COLUMNAS_IGUALES.append(col)
    else: #Implementación opcional memoria
        if globals.MEMORIA_DIFERENTES:
            if nueva_puntuacion == value:
                globals.COLUMNAS_IGUALES.append(col)

        if globals.MEMORIA_DIFERENTES: #Implementación opcional memoria
            if len(globals.COLUMNAS_IGUALES) > 1:
                mejorColumna = random.choice(globals.COLUMNAS_IGUALES)
                mejorFila = getPrimerCeroEnLaColumna(tablero,mejorColumna)
                globals.COLUMNAS_IGUALES.clear()

return mejorColumna,mejorFila,value
```

Figura 19: Jugadas diferentes

Tal y como se observa en la figura 19, he añadido las lines marcadas con el recuadro rojo. Para activar o desactivar esta función extra de la memoria he creado un archivo nuevo llamado 'globals' donde tengo una variable global llamada 'MEMORIA DIFERENTES' para activar o descativar esta función extra cuando plazca. Nota: estas lines de código se añaden tanto en la condición del minimazing como el maximazing.

3. AlfaBeta

3.1. ¿Dónde se introduce la poda alfa-beta en tu algoritmo minimax?

La poda en mi algoritmo minimax se produce al terminar cada iteración del bucle que recorre todas las columnas sin llenar.

```

def AlfaBeta(tablero, profundidad, maximizingPlayer, alfa, beta):
    columnasSinLlenar = obtenerColumnasSinLlenar(tablero)
    esTerminal = esNodoTerminal(tablero)

    if profundidad == 0 or esTerminal:
        if esTerminal:
            if tablero.cuatroEnRaya() == PIEZA_AI:
                return (None, None, 1000000)
            elif tablero.cuatroEnRaya() == PIEZA_JUGADOR:
                return (None, None, -1000000)
            else:
                return (None, None, 0)
        else:
            return (None, None, verPuntuacion_de_esto_tablero(tablero, PIEZA_AI))

    if maximizingPlayer:
        value = -math.inf
        mejorColumna = random.choice(columnasSinLlenar)
        mejorFila = getPrimerCeroEnLaColumna(tablero, mejorColumna)

        for col in columnasSinLlenar:
            fila = getPrimerCeroEnLaColumna(tablero, col)
            aux_tablero = Tablero(tablero)
            aux_tablero.setCelda(fila, col, PIEZA_AI)
            nueva_puntuacion = AlfaBeta(aux_tablero, profundidad-1, False, alfa, beta)[2]

            if nueva_puntuacion > value:
                value = nueva_puntuacion
                mejorColumna = col
                mejorFila = fila

            # PODA ALFA BETA
            alfa = max(alfa, value)
            if alfa >= beta:
                break

        return mejorColumna, mejorFila, value
    else: #minimizingPlayer
        value = math.inf
        mejorColumna = random.choice(columnasSinLlenar)
        mejorFila = getPrimerCeroEnLaColumna(tablero, mejorColumna)

        for col in columnasSinLlenar:
            fila = getPrimerCeroEnLaColumna(tablero, col)
            aux_tablero = Tablero(tablero)
            aux_tablero.setCelda(fila, col, PIEZA_JUGADOR)
            nueva_puntuacion = AlfaBeta(aux_tablero, profundidad-1, True, alfa, beta)[2]

            if nueva_puntuacion < value:
                value = nueva_puntuacion
                mejorColumna = col
                mejorFila = fila

            # PODA ALFA BETA
            beta = min(beta, value)
            if alfa >= beta:
                break

        return mejorColumna, mejorFila, value

```

Figura 20: Poda Alfa-Beta

El problema de minimax es que a partir de la profundidad 4, el ordenador tiene que procesar demasiadas iteraciones de la recursión del minimax en todas sus ramas. Esto lo soluciona el alfa-beta, permitiéndonos ir a profundidades mayores que la 4. Alfa-beta elimina muchos de los nodos que no son necesarios recorrer porque ya no se puede 'mejorar' el resultado obtenido en otro nodo por lo que ir por ese camino es innecesario y 'poda' ese camino.

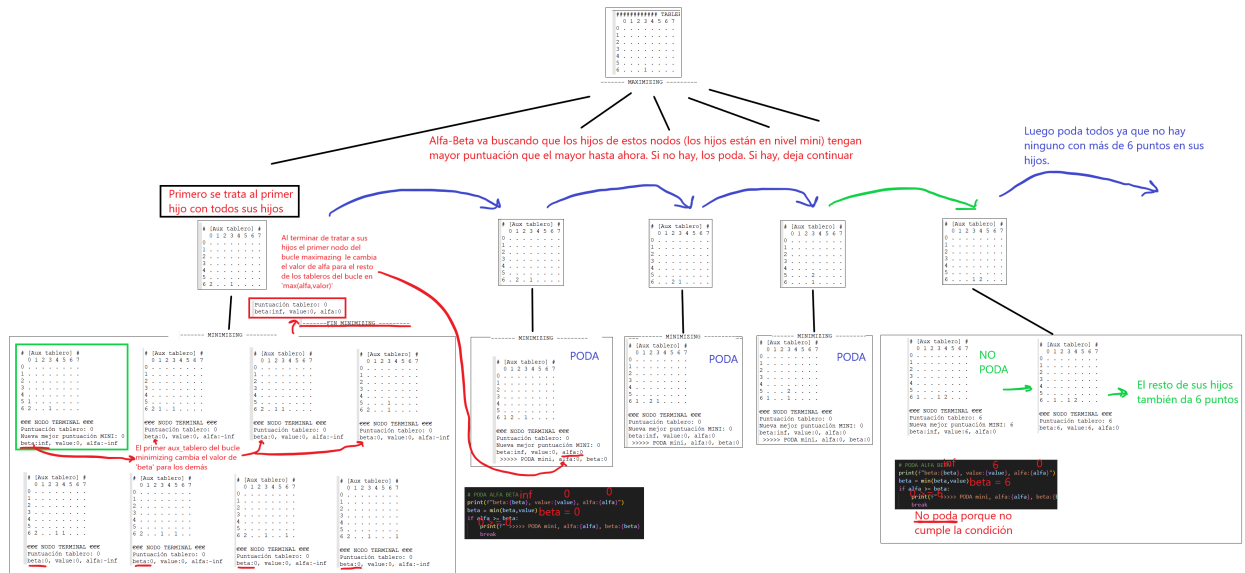


Figura 21: Explicación visual alfa-beta

3.2. ¿Hasta qué nivel de profundidad juega tu algoritmo de manera razonable?

Probando varias partidas el algoritmo alfa-beta juega de manera razonable hasta el nivel 5. En el 6 aún se puede jugar pero tarda un rato en procesar todo, pero se va haciendo más rápido a medida que quedan menos casillas vacías en tablero (es decir, a medida que se va acercando el final del juego).

3.3. ¿Cuántos nodos te ahorras al aplicar alfa-beta vs minimax?, ¿Es más rápido que minimax? Muestra ejemplos

Para ello, en igualdad de condiciones realizo las pruebas necesarias en la misma jugada aplicando minimax y luego alfabeta para ver los tiempos y los nodos que se han empleado para el algoritmo en esa jugada:

```
main.py
globals.CONT_NODOS_MINIMAX = 0
t_minimax_inicio = time.time()
col,row,minimax_score = minimax(tablero,PROFUNDIDAD_MINIMAX,True)
t_minimax_fin = time.time()
print(f"Tiempo_Minimax: {t_minimax_fin-t_minimax_inicio}")
print(f"Total nodos recorridos MiniMax: {globals.CONT_NODOS_MINIMAX}")

globals.CONT_NODOS_ALFABETA = 0
t_alfabeta_inicio = time.time()
col,row,minimax_score = AlfaBeta(tablero,PROFUNDIDAD_ALFABETA,True, -math.inf, math.inf)
t_alfabeta_fin = time.time()
print(f"Tiempo_AlfaBeta: {t_alfabeta_fin-t_alfabeta_inicio}")
print(f"Total nodos recorridos MiniMax: {globals.CONT_NODOS_ALFABETA}")

algoritmo.py
for col in columnasSinllenar:
    fila = getPrimerCeroEnLaColumna(tablero,col)
    aux_tablero = Tablero(tablero)
    globals.CONT_NODOS_ALFABETA = globals.CONT_NODOS_ALFABETA + 1

CONT_NODOS_MINIMAX = 0
CONT_NODOS_ALFABETA = 0

PROFUNDIDAD_MINIMAX = 3
PROFUNDIDAD_ALFABETA = 3
```

Figura 22: MiniMax vs AlfaBeta [configuración]

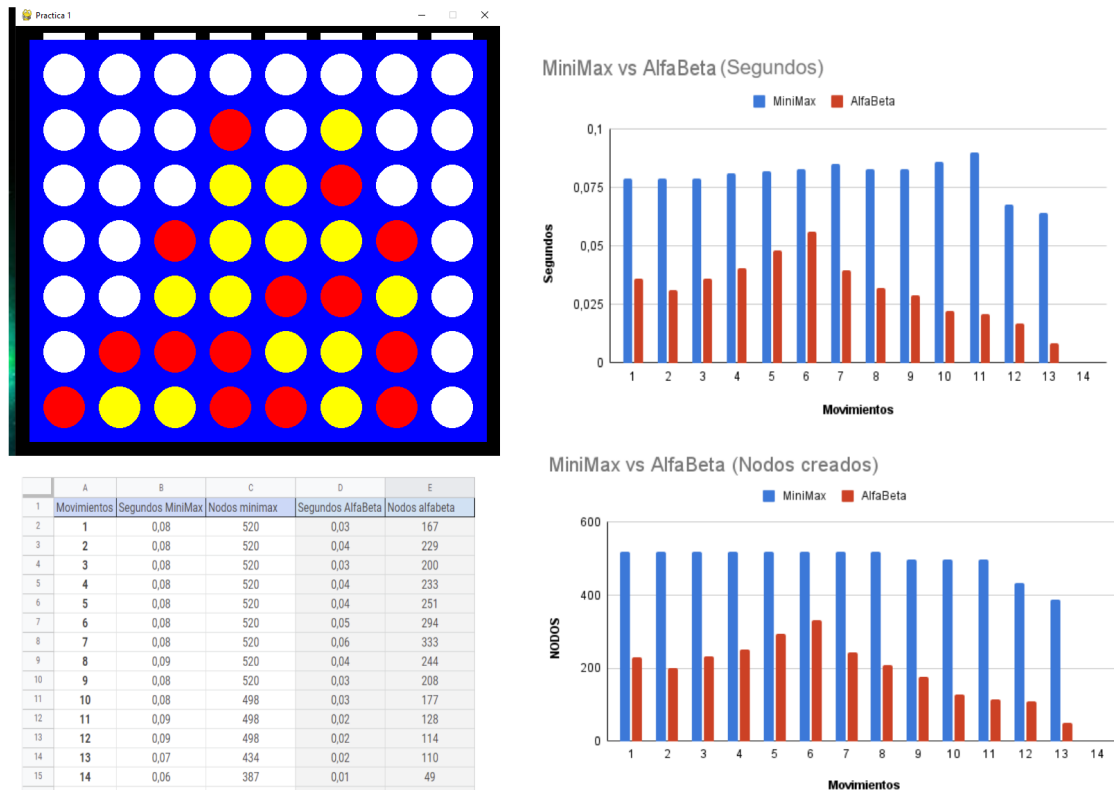


Figura 23: MiniMax vs AlfaBeta [resultados]

Una vez obtenidos los datos de una partida realizada de minimax vs alfabeta (en igualdad de condiciones) se puede observar que el alfabeta recorre muchos menos nodos y es un 60 por ciento de media más rápido que minimax (basado en los datos obtenidos en la figura 23).

4. Comparación de funciones de evaluación

He realizado distintas partidas de AI vs AI para poder comprobar los resultados de las mismas. Para preparar el código que juegue máquina vs máquina (en vez de jugador vs máquina) he tenido que añadir fuera del bucle `for event in pygame.event.get()` el código de máquina vs máquina (en el cual solo si la variable global `JUGADOR-VS-MAQUINA` está a `False`). A continuación expondré las diferentes partidas jugadas con la información de 'quien juega, si IA 1 o IA 2', 'El algoritmo empleado, si AlfaBeta o MiniMax', 'La profundidad PF', 'Y si es GANADOR uno o el otro'

Las conclusiones que se obtienen al enfrentar a 2 máquinas entre si es que a pesar de tener las mismas características respecto al algoritmo usado, siempre ganará la máquina que empieza. Otra de las conclusiones es que en las mismas condiciones, a pesar de que la máquina 1 empiece, si coloca la ficha en una columna u otra influirá en si al final gana o no.

```

while not game_over:
    for event in pygame.event.get(): ...

    if globals.JUGADOR_VS_MAQUINA == False:
        if once: #para que se dibuje el mapa al principio
            once = False
            dibujarTablero(tablero, reloj, screen)

        col1, row1, minimax_score1 = AlfaBeta(tablero, 4, True, -math.inf, math.inf)

        #PARA TESTEO COLOCANDO EN COLUMNAS SUCEASIVAS
        if once_columna:
            tablero.setCelda(6, 7, 1)
            once_columna = False
        else:
            tablero.setCelda(row1, col1, 1)

    print(tablero)
    if tablero.cuatroEnRaya() == 1: #Compruebo si he ganado
        game_over = True
        print("gana maquina 1")
        print('\033[91m' + 'Tablero Ganador maquina 1:' + '\033[0m')
        print(tablero)
    else:
        pygame.time.wait(100)
        #col2, row2, minimax_score2 = miniMax(tablero, PROFUNDIDAD_ALFABETA, True)
        col2, row2, minimax_score2 = AlfaBeta(tablero, 4, True, -math.inf, math.inf)

        tablero.setCelda(row2, col2, 2)
        print(tablero)
        if tablero.cuatroEnRaya() == 2: #Compruebo si he ganado
            game_over = True
            print("gana maquina 2")
            print('\033[91m' + 'Tablero Ganador maquina 2:' + '\033[0m')
            print(tablero)

```

Figura 24: main IA vs IA

- (IA 1: AlfaBeta(PF: 3) GANADOR) vs (IA 2: MiniMax(PF: 3))

	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

→

	0	1	2	3	4	5	6	7
0	2	.	1	.	1	2	.	2
1	1	.	2	.	2	2	.	1
2	2	.	1	.	1	2	.	2
3	1	.	2	.	1	1	.	1
4	2	.	1	.	2	2	1	2
5	1	1	1	.	2	1	2	1
6	1	1	2	1	1	2	2	2

Figura 25: AlfaBeta vs MiniMax. Caso 1

- (IA 1: MiniMax(PF: 3) GANADOR) vs (IA 2: AlfaBeta(PF: 3))


	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

→

	0	1	2	3	4	5	6	7
0	2	2	.	2	1	.	.	1
1	1	1	.	2	2	.	.	1
2	2	2	.	2	1	.	.	2
3	1	2	.	1	1	.	1	1
4	2	2	1	2	2	.	1	2
5	1	1	2	1	2	.	1	1
6	2	2	1	2	1	1	2	1

Figura 26: MiniMax vs AlfaBeta. Caso 2

- (IA 1: AlfaBeta(PF: 3) GANADOR) vs (IA 2: AlfaBeta(PF: 3))




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

	0	1	2	3	4	5	6	7
0	1	2	2	.	2	1	.	2
1	2	1	1	.	1	2	.	1
2	1	2	2	.	2	1	.	1
3	2	1	1	1	1	2	.	2
4	1	1	2	2	1	1	.	1
5	2	2	1	1	2	1	.	2
6	1	2	2	1	1	2	2	2

Figura 27: AlfaBeta vs AlfaBeta. Caso 3

- (IA 1: AlfaBeta(PF: 2)) vs (IA 2: AlfaBeta(PF: 5) GANADOR)




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

	0	1	2	3	4	5	6	7
0	.	.	2	1	1	.	.	.
1	.	.	2	1	1	.	.	.
2	.	.	1	1	2	.	.	.
3	.	.	2	2	2	2	.	.
4	.	.	2	1	2	1	.	.
5	.	1	2	1	1	1	.	.
6	2	2	1	2	1	2	.	.

Figura 28: AlfaBeta vs AlfaBeta. Caso 4

- (IA 1: AlfaBeta(PF: 3) GANADOR) vs (IA 2: AlfaBeta(PF: 5))




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

	0	1	2	3	4	5	6	7
0	1	2	2	.	2	1	.	2
1	2	1	1	.	1	2	.	1
2	1	2	2	.	2	1	.	1
3	2	1	1	1	1	2	.	2
4	1	1	2	2	1	1	.	1
5	2	2	1	1	2	1	.	2
6	1	2	2	1	1	2	2	2

Figura 29: AlfaBeta vs AlfaBeta. Caso 5

- (IA 1: AlfaBeta(PF: 4),Empieza Col:0, GANADOR) vs (IA 2: AlfaBeta(PF: 4))




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1

	0	1	2	3	4	5	6	7
0	2	1	1	2	2	1	.	1
1	1	2	2	1	1	2	.	2
2	2	1	1	2	2	2	.	1
3	1	2	2	1	1	1	.	2
4	2	1	1	2	2	2	.	1
5	1	2	2	1	1	1	1	2
6	1	2	1	1	2	2	2	1

Figura 30: AlfaBeta vs AlfaBeta. Caso Columna 0

- (IA 1: AlfaBeta(PF: 4),Empieza Col:1) vs (IA 2: AlfaBeta(PF: 4) GANADOR)




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	.	1

	0	1	2	3	4	5	6	7
0	2	1	1	.	1	1	.	.
1	1	2	2	.	2	2	.	.
2	2	1	1	.	1	1	.	.
3	1	1	2	.	1	2	2	.
4	2	2	1	.	2	1	2	1
5	2	1	2	1	1	2	2	2
6	2	1	1	2	2	1	2	1

Figura 31: AlfaBeta vs AlfaBeta. Caso Columna 1

- (IA 1: AlfaBeta(PF: 4),Empieza Col:2) vs (IA 2: AlfaBeta(PF: 4) GANADOR)




	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	.	.	1

	0	1	2	3	4	5	6	7
0	2	.	2	.	2	2	.	.
1	1	.	1	.	1	1	.	.
2	1	.	2	.	2	1	.	.
3	2	.	1	.	1	2	.	.
4	2	2	1	.	2	1	2	.
5	1	1	2	.	1	2	2	.
6	1	2	1	2	2	1	1	1

Figura 32: AlfaBeta vs AlfaBeta. Caso Columna 2

- (IA 1: AlfaBeta(PF: 4),Empieza Col:3, GANADOR) vs (IA 2: AlfaBeta(PF: 4))



	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	.	.	.	1

	0	1	2	3	4	5	6	7
0	1	.	.	.	2	1	.	.
1	2	.	.	.	2	2	2	.
2	1	.	.	.	2	1	1	.
3	1	.	.	.	1	1	2	.
4	2	.	1	2	2	1	.	1
5	1	1	1	2	1	2	.	2
6	2	2	1	1	2	1	.	2

Figura 33: AlfaBeta vs AlfaBeta. Caso Columna 3

- (IA 1: AlfaBeta(PF: 4),Empieza Col:4, GANADOR) vs (IA 2: AlfaBeta(PF: 4))

	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

	0	1	2	3	4	5	6	7
0	2	2	2	.	2	2	.	1
1	1	1	1	.	1	1	.	2
2	2	2	2	1	.	2	1	.
3	1	1	2	.	2	2	.	2
4	2	2	1	.	1	1	.	1
5	1	1	2	1	1	2	.	2
6	1	2	2	2	1	1	2	1

Figura 34: AlfaBeta vs AlfaBeta. Caso Columna 4

- (IA 1: AlfaBeta(PF: 4),Empieza Col:5, GANADOR) vs (IA 2: AlfaBeta(PF: 4))

	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.	.

	0	1	2	3	4	5	6	7
0	2	1	2	2	2	1	.	1
1	1	2	2	1	1	1	.	1
2	2	2	1	1	2	2	1	.
3	1	1	2	2	1	2	.	1
4	2	1	1	2	2	2	.	2
5	1	2	2	1	1	1	1	1
6	2	1	1	2	2	1	2	2

Figura 35: AlfaBeta vs AlfaBeta. Caso Columna 5


- (IA 1: AlfaBeta(PF: 4),Empieza Col:6) vs (IA 2: AlfaBeta(PF: 4) GANADOR)

	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1	.	.

	0	1	2	3	4	5	6	7
0	2	2	1	.	1	.	1	.
1	1	1	2	.	1	.	2	2
2	2	2	1	.	2	1	2	2
3	1	1	2	.	1	2	2	1
4	1	2	2	.	2	1	1	1
5	2	1	1	.	1	2	1	2
6	2	2	1	.	2	1	1	2

Figura 36: AlfaBeta vs AlfaBeta. Caso Columna 6

- (IA 1: AlfaBeta(PF: 4),Empieza Col:7,GANADOR) vs (IA 2: AlfaBeta(PF: 4))



	0	1	2	3	4	5	6	7
0
1
2
3
4
5
6	1

	0	1	2	3	4	5	6	7
0	1	.	2	1	2	1	.	.
1	1	.	2	1	1	2	.	.
2	2	1	2	2	2	1	.	2
3	2	2	1	2	1	1	.	1
4	1	2	1	1	2	2	.	2
5	2	1	2	2	1	1	.	1
6	1	1	2	1	2	2	.	1

Figura 37: AlfaBeta vs AlfaBeta. Caso Columna 7