MiniMax

Práctica 1

Sistemas Inteligentes

Noel Martínez Pomares
48771960T TURNO DE PRÁCTICAS -> JUEVES 9-11

Índice

1. Introducción

En esta primera práctica el alumno debe desarrollar un algoritmo de búsqueda para el juego Conecta-4. En este juego el jugador va colocando fichas sobre un tablero vertical para intentar conseguir 4 fichas en línea; ya sea en horizontal, vertical o diagonal.

El objetivo es diseñar una función de evaluación para el juego, estudiar su modo de funcionamiento y por qué interesa colocar la ficha en una posición concreta. La función de evaluación debe devolver una medida numérica de lo buena que es una determinada configuración del tablero para la máquina.

Al ejecutar el juego aparece un tablero preparado para introducir las fichas. El juego está pensado para funcionar en modo máquina-persona, empezando la persona. Para colocar una ficha se debe pulsar con el ratón en la columna en la que se quiere colocar la ficha. Si la columna tiene hueco se colocará una ficha de color rojo en la posición seleccionada. A continuación es el turno de la máquina, que mediante el algoritmo implementado determina la posición óptima y coloca una ficha amarilla en dicha posición.

El juego continúa hasta que uno de los dos consigue colocar 4 fichas en línea en cualquiera de las direcciones o hasta que el tablero se llena.

Al alumno se le otorga un código donde la máquina decide su jugada simplemente barriendo el tablero de izquierda a derecha, buscando la primera columna disponible donde colocar la ficha.

El entorno se ha desarrollado en Python usando la librería "pygame", que permite la creación de videojuegos en dos dimensiones y permite programar la parte multimedia de forma sencilla.

2. Función de evaluación

La función de evaluación es usada por la función MiniMax para analizar el tablero, otorgando a este un valor que depende de la cantidad de jugadas que hay en juego, dando un valor positivo a las jugadas de la máquina y dando un valor negativo a las jugadas del jugador.

La inteligencia de la máquina depende totalmente de la función de evaluación, ya que esta es la que le proporciona al MiniMax, el valor de cuan buena es la posición que se está evaluando, y si la evaluación es incorrecta, nunca podrá jugar de manera inteligente.

Implementación de la función de evaluación escogida.

La función se basa en recorrer todo el tablero, posición por posición e ir comprobando si las posiciones adyacentes a dichas posiciones analizadas son del mismo bando ellas.

Para ello se han usado 4 funciones distintas, la primera obtiene el valor del tablero en cuanto a las fichas en posición horizontal, la segunda función obtiene el valor del tablero en cuanto a las fichas en posición vertical, la tercera función obtiene el valor del tablero en cuanto a las fichas en posición diagonal, y la última función crea un bucle donde recorre junto a dos iteradores (uno para las columnas y otro para las filas) todo el tablero, va llamando a las 3 funciones definidas previamente y va sumando al valor total, los valores que vayan devolviendo dichas funciones.

```python
# Función de evaluación
def evaluarJugada(self):
    i = 0
    casilla = 0
    valor = 0

    while i < self.getAlto():
        j = 0
        while j < self.getAncho():
            casilla = self.getCelda(i,j)
            if casilla != 0:
                # HORIZONTAL
                valor += self.horizontal(i,j,casilla)

                # VERTICAL
                valor += self.vertical(i,j,casilla)

                # DIAGONAL
                valor += self.diagonal(i,j,casilla)
            j += 1
        i += 1
    return valor
```

Para la creación de la función, se ha tomado como referencia la función proporcionada por el centro, que se encarga de comprobar si hay 4 fichas en fila y devolver el bando del ganador. Tomando como referencia esto, se ha usado el mismo "modus operandi"; cambiando que devuelva el valor del tablero, en vez del bando del ganador.

La función de evaluación implementada tiene una complejidad temporal de n^2, ya que consta de dos bucles while, que se ejecutan n veces, siendo n en el caso del primero, la cantidad de filas que hay; y en el caso del segundo, n es la cantidad de columnas que hay. Las demás funciones de valuación descartadas tienen una complejidad n^2, la usar los mismos bucles.

Explicación puntuación de casos base elegidos.

He pasado por casi 7 funciones de evaluación diferentes; algunas muy similares y otras muy diferentes, hasta llegar a la función de evaluación implementada finalmente. Cuando llegaba a una función de evaluación buena, la máquina decidía dejar de jugar cuando sabía que el jugador ya le iba a ganar; esto lo pude solucionar dando más valor a las jugadas del jugador que a las jugadas de la máquina.

Al principio, le daba casi 10 veces más de valor, y por eso la máquina dejaba de jugar, porque veía inalcanzable llegar hasta la puntuación del jugador. Hasta que en una prueba decidí ponerles a las jugadas del jugador, la mitad más de lo que valen las jugadas de la máquina; es decir, si la jugada de la máquina vale 1000, la del jugador vale 1500.

Casos base en Horizontal.

Se comprueba hacia la derecha para jugadas de 2 fichas; j positiva. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia la derecha (j+1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (j+2) está vacía, si debajo de esa posición vacía no está vacío (i+1,j+2), y si la posición siguiente a la posición vacía es del mismo bando (j+3); esta jugada tiene una puntuación de 1000-1500.
- Se comprueba si la siguiente posición (j+2) está vacía, si debajo de esa posición vacía no está vacío (i+1,j+2); esta jugada tiene una puntuación de 100-150.
- Si no es ninguno de estos casos se le da una puntuación de 10-15.

```
if (j+1) < self.getAncho():
    if self.getCelda(i,j+1) == casilla:
        if self.getCelda(i,j+2) == 0 and self.getCelda(i,j+3) == casilla and self.getCelda(i+1,j+2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i,j+2) == 0 and self.getCelda(i+1,j+2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

1100
```

Luego comprobamos lo mismo pero hacía la izquierda, la j negativa.

```
if (j-1) >= 0:
    if self.getCelda(i,j-1) == casilla:
        if self.getCelda(i,j-2) == 0 and self.getCelda(i,j-3) == casilla and self.getCelda(i+1,j-2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i,j-2) == 0 and self.getCelda(i+1,j-2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

1100
```

Se comprueba hacia la derecha para jugadas de 3 fichas; j positiva. Teniendo una ficha en una posición se comprueba si las siguientes 2 fichas hacia la derecha (j+1 y j+2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (j+3) de esa última posición está vacía y si debajo de esa posición vacía no está vacío (i+1,j+3); esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

```
if (j+2) < self.getAncho():
    if self.getCelda(i,j+1) == casilla and self.getCelda(i,j+2) == casilla:
        if self.getCelda(i,j+3) == 0 and self.getCelda(i+1,j+3) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

1230
```

Luego comprobamos lo mismo pero hacía la izquierda, la j negativa.

```
if (j-2) >= 0:
    if self.getCelda(i,j-1) == casilla and self.getCelda(i,j-2) == casilla:
        if self.getCelda(i,j-3) == 0 and self.getCelda(i+1,j-3) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

1230
```

Por último se comprueba si hay una jugada de 4 fichas en línea de manera horizontal, lo que significaría que el bando correspondiente a esa jugada de fichas será el bando ganador de la partida. Se le dará una puntuación de 10000-15000.

```
if (j+3) < self.getAncho():
    if self.getCelda(i,j+1) == casilla and self.getCelda(i,j+2) == casilla and self.getCelda(i,j+3) == casilla:
        if casilla == 2:
            valor += 10000
        else:
            valor -= 15000
```

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

12440
```

Casos base en Vertical.

Se comprueba hacia arriba para jugadas de 2 fichas; i negativa. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia arriba (i-1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-2) está vacía; esta jugada tiene una puntuación de 100-150.
- Sino se le da una puntuación de 10-15.

```
if (i-1) >= 0:
    if self.getCelda(i-1,j) == casilla:
        if self.getCelda(i-2,j) == 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

20
```

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

110
```

Luego comprobamos pero hacía abajo, la i positiva. Esta vez no tenemos que hacer más de una comprobación, pues hacía abajo es imposible que haya un posición vacía. Simplemente se comprueba que la posición de abajo es del mismo bando; si es así, se le da una puntuación de 10-15.

```
if (i+1) < self.getAlto():
    if self.getCelda(i+1,j) == casilla:
        if casilla == 2:
            valor += 10
        else:
            valor -= 15
```

Se comprueba hacia arriba para jugadas de 3 fichas; i negativa. Teniendo una ficha en una posición se comprueba si las siguientes 2 fichas hacia arriba (i-1 y i-2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-3) está vacía; esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

```
if (i-2) >= 0:
    if self.getCelda(i-1,j) == casilla and self.getCelda(i-2,j) == casilla:
        if self.getCelda(i-3,j) == 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

1230
```

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

240
```

Luego comprobamos pero hacía abajo, la i positiva. No tenemos que hacer más de una comprobación, pues hacía abajo es imposible que haya un posición vacía. Simplemente se comprueba que las 2 posiciones de abajo son del mismo bando; si es así, se le da una puntuación de 100-150.

```
if (i+2) < self.getAlto():
    if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla:
        if casilla == 2:
            valor += 100
        else:
            valor -= 150
```

Por último se comprueba si hay una jugada de 4 fichas en línea de manera vertical, lo que significaría que el bando correspondiente a esa jugada de fichas será el bando ganador de la partida. Se le dará una puntuación de 10000-15000.

```
if (i+3) < self.getAlto():
    if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla and self.getCelda(i+3,j) == casilla:
        if casilla == 2:
            valor += 15000
        else:
            valor -= 15000
```

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

11450
```

Casos base en Diagonal.

Se comprueba hacia abajo-derecha para jugadas de 2 fichas; j-i positivas. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia la abajo-derecha (i+1,j+1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i+2,j+2) está vacía, si debajo de esa posición vacía no está vacío (i+3,j+2), y si la posición siguiente a la posición vacía es del mismo bando (i+3,j+3); esta jugada tiene una puntuación de 1000-1500.
- Se comprueba si la siguiente posición (i+2,j+2) está vacía, si debajo de esa posición vacía no está vacío (i+3,j+2); esta jugada tiene una puntuación de 100-150.
- Si no es ninguno de estos casos se le da una puntuación de 10-15.

```
if (j+1) < self.getAncho():
    if self.getCelda(i+1,j+1) == casilla:
        if self.getCelda(i+2,j+2) == 0 and self.getCelda(i+3,j+3) == casilla and self.getCelda(i+3,j+2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i+2,j+2) == 0 and self.getCelda(i+3,j+2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

240
```

Se comprueba hacia abajo-izquierda para jugadas de 2 fichas; i positiva y j negativa. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia la abajo-izquierda (i+1,j-1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i+2,j-2) está vacía, si debajo de esa posición vacía no está vacío (i+3,j-2), y si la posición siguiente a la posición vacía es del mismo bando (i+3,j-3); esta jugada tiene una puntuación de 1000-1500.
- Se comprueba si la siguiente posición (i+2,j-2) está vacía, si debajo de esa posición vacía no está vacío (i+3,j-2); esta jugada tiene una puntuación de 100-150.
- Si no es ninguno de estos casos se le da una puntuación de 10-15.

```
if (j-1) >= 0:
    if self.getCelda(i+1,j-1) == casilla:
        if self.getCelda(i+2,j-2) == 0 and self.getCelda(i+3,j-3) == casilla and self.getCelda(i+3,j-2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i+2,j-2) == 0 and self.getCelda(i+3,j-2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

240
```

Se comprueba hacia arriba-izquierda para jugadas de 2 fichas; j-i negativas. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia la arriba-izquierda (i-1,j-1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-2,j-2) está vacía, si debajo de esa posición vacía no está vacío (i-1,j-2), y si la posición siguiente a la posición vacía es del mismo bando (i-3,j-3); esta jugada tiene una puntuación de 1000-1500.
- Se comprueba si la siguiente posición (i-2,j-2) está vacía, si debajo de esa posición vacía no está vacío (i-1,j-2); esta jugada tiene una puntuación de 100-150.
- Si no es ninguno de estos casos se le da una puntuación de 10-15.

```
if (j-1) >= 0:
    if self.getCelda(i-1,j-1) == casilla:
        if self.getCelda(i-2,j-2) == 0 and self.getCelda(i-3,j-3) == casilla and self.getCelda(i-1,j-2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i-2,j-2) == 0 and self.getCelda(i-1,j-2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

-1945
```

Se comprueba hacia arriba-derecha para jugadas de 2 fichas; i negativa y j positiva. Teniendo una ficha en una posición se comprueba si la siguiente ficha hacia la abajo-izquierda (i-1,j+1) es del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-2,j+2) está vacía, si debajo de esa posición vacía no está vacío (i-1,j+2), y si la posición siguiente a la posición vacía es del mismo bando (i-3,j+3); esta jugada tiene una puntuación de 1000-1500.
- Se comprueba si la siguiente posición (i-2,j+2) está vacía, si debajo de esa posición vacía no está vacío (i-1,j+2); esta jugada tiene una puntuación de 100-150.
- Si no es ninguno de estos casos se le da una puntuación de 10-15.

```
if (j+1) < self.getAncho():
    if self.getCelda(i-1,j+1) == casilla:
        if self.getCelda(i-2,j+2) == 0 and self.getCelda(i-3,j+3) == casilla and self.getCelda(i-1,j+2) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        elif self.getCelda(i-2,j+2) == 0 and self.getCelda(i-1,j+2) != 0:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
        else:
            if casilla == 2:
                valor += 10
            else:
                valor -= 15
```

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

365
```

Se comprueba hacia abajo-derecha para jugadas de 3 fichas; j-i positivas. Teniendo una ficha en una posición se comprueba si las siguientes 3 fichas hacia abajo-derecha (i+1,j+1 y i+2,j+2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i+3,j+3) está vacía, si debajo de esa posición vacía no está vacío (i+4,j+3); esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

```
if (j+2) < self.getAncho():
    if self.getCelda(i+1,j+1) == casilla and self.getCelda(i+2,j+2) == casilla:
        if self.getCelda(i+3,j+3) == 0 and self.getCelda(i+4,j+3) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

345
```

Se comprueba hacia abajo-izquierda para jugadas de 3 fichas; i positiva y j negativa. Teniendo una ficha en una posición se comprueba si las siguientes 3 fichas hacia abajo-izquierda (i+1,j-1 y i+2,j-2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i+3,j-3) está vacía, si debajo de esa posición vacía no está vacío (i+4,j-3); esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

```
if self.getCelda(i+1,j-1) == casilla and self.getCelda(i+2,j-2) == casilla:
    if self.getCelda(i+3,j-3) == 0 and self.getCelda(i+4,j-3) != 0:
        if casilla == 2:
            valor += 1000
        else:
            valor -= 1500
    else:
        if casilla == 2:
            valor += 100
        else:
            valor -= 150
```

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

585
```

Se comprueba hacia arriba-izquierda para jugadas de 3 fichas; i-j negativas. Teniendo una ficha en una posición se comprueba si las siguientes 3 fichas hacia arriba-izquierda (i-1,j-1 y i-2,j-2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-3,j-3) está vacía, si debajo de esa posición vacía no está vacío (i-2,j-3); esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

```
if (j-2) >= 0:
    if self.getCelda(i-1,j-1) == casilla and self.getCelda(i-2,j-2) == casilla:
        if self.getCelda(i-3,j-3) == 0 and self.getCelda(i-2,j-3) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

345
```

Se comprueba hacia arriba-derecha para jugadas de 3 fichas; i negativa y j positiva. Teniendo una ficha en una posición se comprueba si las siguientes 3 fichas hacia arriba-derecha (i-1,j+1 y i-2,j+2) son del mismo bando; si es así, hay varias posibilidad:

- Se comprueba si la siguiente posición (i-3,j+3) está vacía, si debajo de esa posición vacía no está vacío (i-2,j+3); esta jugada tiene una puntuación de 1000-1500.
- Sino se le da una puntuación de 100-150.

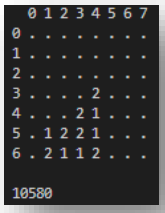
```
if (j+2) < self.getAncho():
    if self.getCelda(i-1,j+1) == casilla and self.getCelda(i-2,j+2) == casilla:
        if self.getCelda(i-3,j+3) == 0 and self.getCelda(i-2,j+3) != 0:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1500
        else:
            if casilla == 2:
                valor += 100
            else:
                valor -= 150
```

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

585
```

Por último se comprueba si hay una jugada de 4 fichas en línea de manera diagonal, lo que significaría que el bando correspondiente a esa jugada de fichas será el bando ganador de la partida. Se le dará una puntuación de 10000-15000.

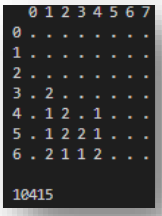
Se comprueba para j negativa:

```
if (j-3) >= 0:
    if self.getCelda(i+1,j-1) == casilla and self.getCelda(i+2,j-2) == casilla and self.getCelda(i+3,j-3) == casilla:
        if casilla == 2:
            valor += 10000
        else:
            valor -= 15000
```

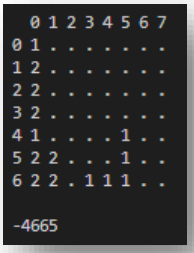


Se comprueba para j positiva:

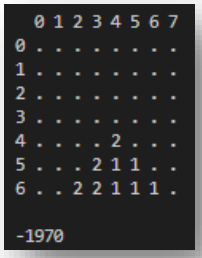
```
if (j+3) < self.getAncho():
    if self.getCelda(i+1,j+1) == casilla and self.getCelda(i+2,j+2) == casilla and self.getCelda(i+3,j+3) == casilla:
        if casilla == 2:
            valor += 10000
        else:
            valor -= 15000
```



Ejemplos de jugadas.



La jugada de 3 del bando máquina, se ve bloqueada por fichas del bando jugador y esto hace que pase de valer 1000 a valer 100.



Hay una jugada de 2 en diagonal del bando jugador que al tener una ficha debajo de la tercera posible posición, cuenta 150 y no 15.

¿Se ha implementado una nueva clase para tratar con el tablero? ¿Por qué?

El código de la función se encuentra en el fichero "tablero.py", ya que esta función trabaja directamente sobre el tablero y no tendría sentido hacerla en el fichero del algoritmo, ya que en el fichero de la clase tablero podemos aprovechar directamente todas sus funciones internas.

La opción de crear una nueva clase para la función de evaluación no ha sido implementada, pues perderíamos rendimiento, teniendo que llamar desde ella a la clase tablero.

Diferentes funciones de evaluación implementadas.

Antes de llegar a la función de evaluación definitiva; que tengo implementada en mi trabajo, he pasado por muchas otras versiones que no han llegado a nada, por el mal funcionamiento o por el mal rendimiento.

Primera función de evaluación descartada

Esta primera función de evaluación era muy parecida a la que está finalmente implementada en el trabajo, pero estaba planteada e implementada de una manera incorrecta, pues solamente comprobaba si la siguiente posición estaba vacía, cuando había 3 fichas conectadas.

Segunda función de evaluación descartada

La idea de esta segunda función de evaluación era comprobar primero si hay hueco para un jugada en concreto y luego comprobar si hay o no fichas conectadas.

Tercera función de evaluación descartada

Esta función de evaluación se basaba en comprobar si el algoritmo funciona de mejor forma si no se hacen comprobaciones de si la siguiente posición es una posición vacía o no.

Cuarta función de evaluación descartada

Esta versión de la función de evaluación se basaba en comprobar si la posición siguiente está vacía o si la posición siguiente es de nuestro equipo, y no del contrario.

3. Ficheros

3.1. main.py

Es el fichero que hay que ejecutar para lanzar el entorno, donde se encuentra el bucle principal del manejo del juego, así como el desarrollo del interfaz gráfico. Comprueba si ha ganado un jugador u otro en cada tirada. Contiene el código necesario para imprimir por pantalla el tablero, para limpiar la pantalla una vez terminado el juego, para actualizar la pantalla cada vez que se coloca una ficha y para cerrar el juego.

Empieza jugando la persona, se comprueba si la posición seleccionada por el jugador es una posición correcta en el tablero; si es así, la ficha se coloca y se llama al método ".cuatroEnRaya()" para comprobar si hay 4 fichas en línea o no. Si después de colocar la persona hay 4 fichas en línea, termina la partida. Si la persona no ha ganado, es el turno de la máquina, que se encarga de llamar a la función que le calcula cual es la posición óptima para su próximo movimiento. Si después de jugar la máquina hay 4 fichas en línea, termina la partida.

Este bucle continua hasta el final de la partida. Al terminar, hay un retardo para no cerrar al instante, después de esto se llamará al código que limpia la pantalla.

3.2. tablero.py

Esta clase permite construir un objeto tablero, así como la matriz que representa las posiciones de las fichas.

El tablero del juego tiene una longitud de 7 alto x 8 ancho y un constructor donde se inicializa la matriz que representa el tablero, ya sea vacía o haciendo una copia del tablero que se le pase como parámetro. La X va hacía la derecha y la Y va hacia abajo.

La clase tiene funciones para devolver el alto y ancho; así como para obtener o cambiar una posición o celda.

La función ".cuatroEnRaya()" comprueba si hay 4 fichas en línea en alguna parte del tablero, recorriendo cada casilla del tablero y comprobando sus fichas adyacentes. Si se da el caso de que efectivamente alguien ha ganado la partida, devolverá 1 si ha ganado el jugador o 2 si ha ganado la máquina. Si nadie ha ganado devuelve 0, significando que nadie ha ganado todavía y hay empate.

La función de evaluación se encuentra en esta clase, ya que trabaja directamente con el tablero, y estando aquí puede aprovechar mejor las funciones de la clase tablero.

```
def getCelda(self, fila, col):
    if col > 7 or fila > 6 or col < 0 or fila < 0:
        return "Fuera del tablero"
    else:
        return self.tablero[fila][col]
```

El único cambio que se ha introducido en el código que nos proporciona el centro, ha sido añadir una comprobación adicional en la función "getCelda()", que comprueba si la celda se sale del tablero o no.

3.3. algoritmo.py

3.3.1. def busca(tablero, columna):

```python
def busca(tablero, columna):
    i = 0
    while i < tablero.getAlto() and tablero.getCelda(i,columna) == 0:
        i += 1
    i -= 1

    return i
```

Recibe el tablero junto a un indicador de columna y devuelve la fila de la primera celda disponible de esa columna del tablero. Esta función es muy útil para las siguientes funciones.

3.3.2. def juega(tablero, posicion):

El objetivo de esta función es cambiar el valor "posición" por la posición que corresponda a la mejor juagada posible para la máquina.

Lo primero que hace la función es inicializar una variable con una cantidad numérica negativa absurdamente grande, que servirá para guardar el mayor valor que devuelva la función MiniMax hasta el momento y; además, servirá para diferenciar la mejor posición de las demás. A continuación comienza un bucle que recorre las 8 columnas del tablero, y con la ayuda de la función ".busca()", comprueba si la primera fila vacía de cada columna es una posición posible para la jugada. Si es posible colocar la ficha en esa fila, se crea un nuevo tablero clonando el tablero de la partida en ese determinado momento y se asigna al tablero una ficha con el valor 2 (ficha del bando máquina) en la posición que corresponda con la fila y la columna en la que se encuentre el bucle en ese momento.

Una vez que está listo el nuevo tablero con la posible mejor posición, es llamada la función que actúa de MiniMax, que devuelve el valor de la mejor jugada de se obtendría al colocar la ficha de la máquina en esa posición recién colocada en el tablero clonado. Si el valor que devuelve la función MiniMax es mayor que el valor que se había inicializado anteriormente como mejor valor, se sustituye por el valor del MiniMax, y se asigna la fila y la columna de esa mejor posición a la posición pasada como parámetro, que inicialmente es [-1,-1].

Este bucle se realiza mientras que haya posibles posiciones donde colocar la ficha de la máquina, y termina con la mejor posición donde la máquina pueda colocar la ficha.

```python
def juega(tablero, posicion):
    mejorJugada = -100000

    for columna in range(0,tablero.getAncho()):
        fila = busca(tablero,columna)

        if fila != -1:
            nuevoTablero = Tablero(tablero)
            nuevoTablero.setCelda(fila,columna,2)
            valorJugada = V(nuevoTablero,0,False,ALPHA,BETA)

            if valorJugada > mejorJugada:
                mejorJugada = valorJugada
                posicion[0] = fila
                posicion[1] = columna
```

4. MiniMax

Explicación detallada del algoritmo MiniMax

El algoritmo MiniMax se basa en usar una estrategia exhaustiva, donde se generan todos los nodos del árbol hasta la profundidad deseada, se evalúan los nodos hoja y se asigna un valor al nodo raíz: si la decisión la toma el jugador MIN, asociar a ese nodo el mínimo de los valores de sus hijos, y el máximo en caso de MAX. El algoritmo calcula el valor de un tablero en un momento determinado de la partida.

En el algoritmo MiniMax, la máquina es MAX y el jugador es MIN. El jugador MAX busca maximizar el juego; conseguir más puntos con su jugada, y el jugador MIN busca todo lo contrario, conseguir la mayor cantidad de puntos negativos con su jugada.

Al empezar la partida elije el jugador (MIN), a partir de ahí la maquina (MAX) empieza a tomar decisiones. Una vez que el jugador coloca su primera ficha, la máquina tiene 8 posiciones diferentes (hijos) para poder colocar su ficha, por eso crea un árbol de una profundidad determinada.

Una vez que esté completamente expandido hasta esa profundidad, la maquina empieza a analizar las decisiones a partir de los nodos hoja hacia arriba, y cuál es el camino que más le conviene, para ello usa la función de evaluación.

La función recibe el tablero con la posible mejor posición para la siguiente jugada de la máquina, la profundidad escogida para desarrollar el árbol hasta dicha profundidad, un valor booleano que indica de quien es el siguiente turno, el valor Alpha y el valor Beta.

Funcionamiento del algoritmo implementado

Lo primero que hace la función es comprobar si nos encontramos en la máxima profundidad que se le haya asignado, esto significaría que el árbol se encuentra desarrollado hasta los nodos hoja y; a su vez, el algoritmo se encuentra en el último paso de la recursión.

A continuación se comprueba de quien es el supuesto siguiente turno de tirada, como en la función ".juega()" la máquina ha hecho su primera posible elección de posición para la ficha y hemos bajado un nivel de profundidad, es el turno del jugador.

El jugador busca el menor valor posible, para ello se inicializa una variable con una cantidad numérica positiva elevada (negativa en el turno de la máquina); que servirá como indicador del mejor valor hasta el momento, y se realiza un bucle que recorre; como en la función ".juega()", todas las columnas buscando la primera fila de dicha columna, gracias a la función ".busca()".

Si la fila que se ha obtenido de esa columna está vacía, se crea un nuevo tablero copiando el tablero anterior y se le asigna a la posición correspondiente a esa fila y a la columna en la que se encuentre el bucle, una ficha del lado del jugador (del lado máquina, en su turno correspondiente).

A continuación, se realiza la recursividad del algoritmo, haciéndose una auto llamada a la función, pasándose como parámetros el nuevo tablero, un nivel de profundidad más y cambiando el valor del booleano para que en la siguiente posible tirada le toque al contrario.

La llamada recursiva se realizará hasta que llegue a la profundidad escogida y se evalúe el valor del tablero con la función de evaluación, esto ocurrirá desde los últimos nodos creados (nodos hojas) hasta el nodo raíz, que se corresponde con la primera posible elección de la maquina en la función ".juega()". La recursividad nos devuelve el valor numérico de la mejor jugada posible si escogemos esa posición; es decir, como de buena es la posible jugada.

El valor devuelto se compara con el mejor valor que se haya inicializado al comienzo del turno y se pone como nuevo mejor valor, el menor de ellos en el caso de que le toque al jugador y el mayor de ellos en el caso de que le toque a la máquina.

Después de seleccionar el nuevo mejor valor, se realiza la poda de ramas con el algoritmo Alfa-Beta, que se explica en el siguiente punto de la memoria. Una vez terminada la poda, se termina la interacción del bucle y se pasa a la siguiente.

La profundidad influye en el algoritmo y en el juego en general, pues al empezar la máquina a evaluar necesita una profundidad en la que la última jugada sea suya (esto ocurre en las profundidades pares), para poder ver más haya que la persona.

Todo esto sirve para que la máquina tome decisiones por ella misma, analizando todas las posibles jugadas y escogiendo la que más le convenga. Si el jugador está a punto de ganar, la máquina se encargará de interrumpir su victoria, a la vez que ella busca la suya.

```python
def V(tablero, profundidad, MAX, alpha, beta):
    if profundidad == PROFUNDIDAD:
        return tablero.evaluarJugada()
    else:
        if MAX:
            mejorValor = -1000000

            for columna in range(0,tablero.getAncho()):
                fila = busca(tablero,columna)

                if fila != -1:
                    nuevoTablero = Tablero(tablero)
                    nuevoTablero.setCelda(fila,columna,2)
                    valor = V(nuevoTablero,profundidad+1,False,alpha,beta)
                    mejorValor = max(mejorValor,valor)
                    alpha = max(alpha,mejorValor)

                    if beta <= alpha:
                        break

            return mejorValor
```

```python
        else:
            mejorValor = 1000000

            for columna in range(0,tablero.getAncho()):
                fila = busca(tablero,columna)

                if fila != -1:
                    nuevoTablero = Tablero(tablero)
                    nuevoTablero.setCelda(fila,columna,1)
                    valor = V(nuevoTablero,profundidad+1,True,alpha,beta)
                    mejorValor = min(mejorValor,valor)
                    beta = min(beta,mejorValor)

                    if beta <= alpha:
                        break

            return mejorValor
```

¿Que devuelve tu implementación de MiniMax? ¿Cómo accedes a la última jugada?

Mi algoritmo MiniMax está implementado de tal forma que devuelve el valor de la mejor posible jugada; este valor se lo devuelve a la función ".juega()", y ella tiene que determinar cuál es el valor más alto de todos los valores que le proporciona el MiniMax. Esto lo hace para que la máquina haga el mejor movimiento posible.

MiniMax accede a la última jugada gracias a la recursión y gracias a saber; en cada momento, en que profundidad se encuentra. El algoritmo accede a la última jugada cuando llega a la profundidad determinada; en este momento se encuentra en los nodos hoja del árbol, y en último paso de la recursión.

```
if profundidad == PROFUNDIDAD:
    return tablero.evaluarJugada()
```

¿Hasta qué nivel puedes bajar en tu implementación de MiniMax?

Mi implementación de MiniMax baja hasta profundidad 6; sin usar la poda Alpha-Beta, jugando de manera "rápida", haciendo tiempos de entre 50 y 60 segundos de espera entre turno y turno. A mayor profundidad el algoritmo genera demasiadas posibles y jugadas, y extiende demasiado el árbol de búsqueda, en estos casos mi algoritmo tarda demasiado como para poder jugar de manera razonable.

¿Hasta qué nivel de profundidad juega tú algoritmo de manera razonable?

Lo primero a comentar es que en mi implementación, el algoritmo MiniMax comienza en la profundidad 1, pues se llega a esta profundidad en la función ".juega()".

La mejor profundidad (la óptima) para usar en el algoritmo implementado; sin usar la poda Alpha-Beta, es la profundidad 4; en esta profundidad la máquina juega de manera eficiente, coherente y con un rendimiento bastante bueno. Tapando todas nuestras jugadas y; a su vez, jugando de manera agresiva para ganar. En la profundidad 6 también juega excelente, pero los tiempos rondan los 15-30 segundos de espera; mientras que la profundidad 4 no llega ni a 3 segundos de espera.

En el apartado de "Resultados" se muestra la diferencia de tiempos y nodos generados, que hay entre las diferentes profundidades.

¿MiniMax siempre hace la misma jugada para un determinado tablero?

El algoritmo MiniMax siempre hace la misma juagada para un determinado tablero, ya que siempre usa su función de evaluación de la misma manera. Para intentar cambiar esto, podemos obligarle a poner la primera ficha en una posición determinada diferente en cada partida o podemos hacer que no realice siempre la misma jugada para nodos con F(N) iguales.

5. Alpha-Beta

La poda Alfa-Beta realiza la poda de las ramas (jugadas) que no llevan a obtener una mejor solución que la que ya hay hasta el momento. Para conseguir esto se inicializan dos valores absurdamente elevados: Alpha; con valor negativo, y Beta; con valor positivo. Estos dos valores son pasados como parámetros a la función MiniMax desde su llamada en la función ".juega()".

¿Dónde se introduce la poda Alfa-Beta en tu algoritmo MiniMax?

La poda Alpha-Beta se introduce después de seleccionar el nuevo mejor valor; se realiza la poda de ramas con el algoritmo Alpha-Beta, que asigna a la variable Beta pasada como parámetro el menor valor entre dicha variable y el mejor valor seleccionado, en el caso del jugador. En el caso de la máquina, se usa la variable Alpha y se le asigna el mayor de los valores.

En los dos casos si Beta es menos o igual que Alpha, se termina el bucle, indicando que la máquina o jugador no debe seguir desarrollando o evaluando el árbol por esa posible jugada.

```python
if MAX:
    mejorValor = -1000000

    for columna in range(0,tablero.getAncho()):
        fila = busca(tablero,columna)

        if fila != -1:
            nuevoTablero = Tablero(tablero)
            nuevoTablero.setCelda(fila,columna,2)
            valor = V(nuevoTablero,profundidad+1,False,alpha,beta)
            mejorValor = max(mejorValor,valor)
            alpha = max(alpha,mejorValor)

            if beta <= alpha:
                break

    return mejorValor
```

```python
else:
    mejorValor = 1000000

    for columna in range(0,tablero.getAncho()):
        fila = busca(tablero,columna)

        if fila != -1:
            nuevoTablero = Tablero(tablero)
            nuevoTablero.setCelda(fila,columna,1)
            valor = V(nuevoTablero,profundidad+1,True,alpha,beta)
            mejorValor = min(mejorValor,valor)
            beta = min(beta,mejorValor)

            if beta <= alpha:
                break

    return mejorValor
```

¿Cuántos nodos te ahorras al aplicar Alpha-Beta vs MiniMax? ¿Es más rápido que MiniMax?

Usando la poda Alfa-Beta podemos ahorrar hasta 3 veces menos de nodos evaluados, y hasta la mitad de tiempo de ejecución en cada jugada. Esto ya nos indica que merece mucho la pena implementar la poda, pues si queremos bajar de profundidad debemos tenerla implementada.

Esta diferencia entre el algoritmo MiniMax sin y con la poda Alpha-Beta, se explica con ejemplos en la sección de experimentación, que se encuentra en el siguiente punto de la memoria.

¿Hasta qué nivel de profundidad juega tú algoritmo de manera razonable?

Mi algoritmo MiniMax usando la poda Alpha-Beta, juega de manera razonable hasta la profundidad 6; pero puede llegar a jugar hasta en el profundidad 8.

6. Experimentación

6.1. Comprobar función de evaluación

En este primer experimento he comprobado si mi función de evaluación hace de forma correcta el puntuaje. Para ello he usado el fichero que nos proporciona el centro, donde hay varias partidas recreadas, y añadido un par de "print" para conocer que valores calcula.

Una vez impresos los valores, he calculado personalmente los valores de las jugadas, y ciertamente coinciden todos los valores. Por lo que puedo saber con certeza que mi función de evaluación es correcta.

main.py | algoritmo.py | varios tableros.py | tablero.py

P1Conecta4Plantilla > varios tableros.py > ...

```
t1= Tablero(None)
t1.setCelda(max_fil,0,p2)
t1.setCelda(max_fil-1,0,p2)
t1.setCelda(max_fil-2,0,p2)

t1.setCelda(max_fil,2,p1)
t1.setCelda(max_fil,3,p1)

print(t1)
print(t1.evaluarJugada()) #EXPERIMENTO 1
print ("Tableros")

t2= Tablero(None)
t2.setCelda(max_fil,0,p2)
t2.setCelda(max_fil-1,0,p2)
t2.setCelda(max_fil-2,0,p2)

t2.setCelda(max_fil,1,p1)
t2.setCelda(max_fil-1,1,p1)

t2.setCelda(max_fil,5,p1)

print(t2)
print(t2.evaluarJugada()) #EXPERIMENTO 1
```

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

-4665
```

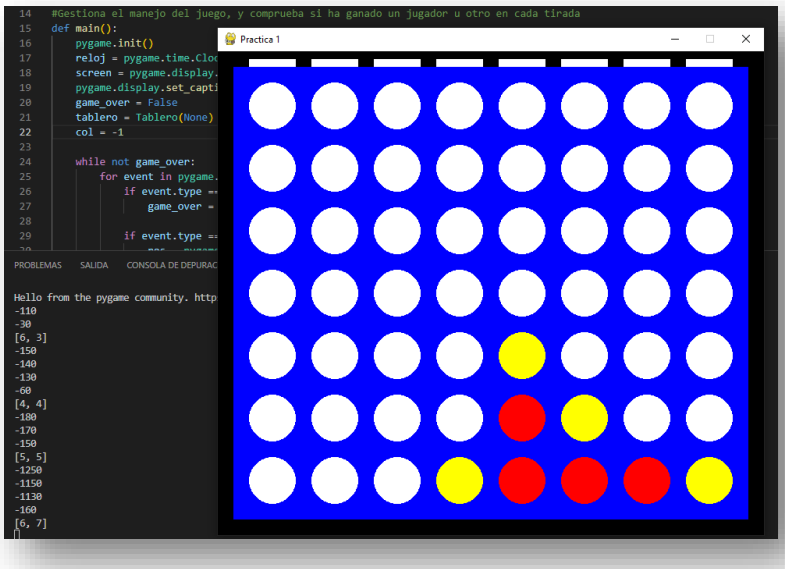
```
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 2 1 . 1 1 . . .

-1650
```

6.2. Comprobar posición y valor correctos

Este experimento lo realicé al comienzo de la implementación del algoritmo, me sirvió para conocer si en mi función ".juega()" se actualizaba la variable "mejorJugada" con un valor mayor que el anterior, esto significaría que el algoritmo MiniMax estaba funcionando. Además de comprobar si la posición que se imprime es la misma que se coloca en el tablero.

El código añadido para la prueba viene indicado como "EXPERIMENTO 2".

```
def juega(tablero, posicion):
    mejorJugada = -100000

    for columna in range(0,tablero.getAncho()):
        fila = busca(tablero,columna)

        if fila != -1:
            nuevoTablero = Tablero(tablero)
            nuevoTablero.setCelda(fila,columna,2)
            valorJugada = V(nuevoTablero,0,False,ALPHA,BETA)

            if valorJugada > mejorJugada:
                mejorJugada = valorJugada
                posicion[0] = fila
                posicion[1] = columna
                print(mejorJugada)  #EXPERIMENTO 2

    print(posicion) #EXPERIMENTO 2
```

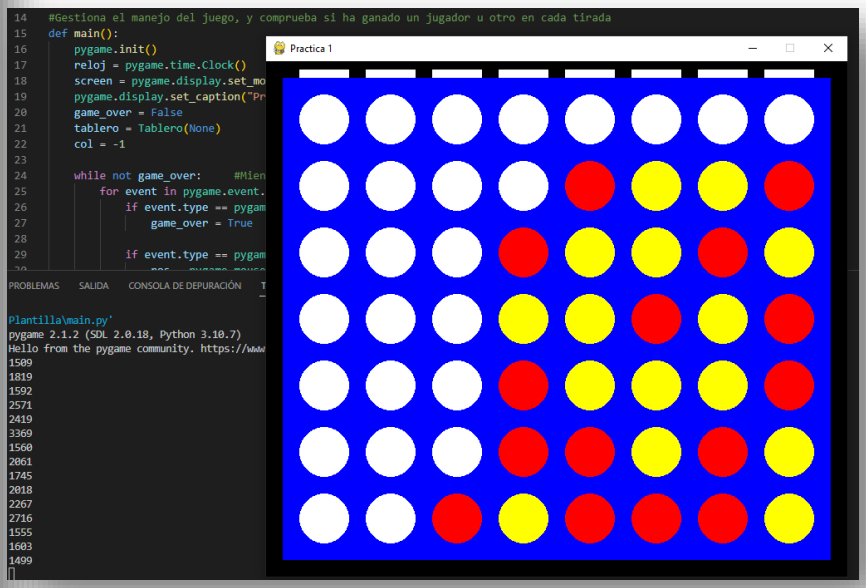


6.3. Cantidad evaluados

En esta tercera prueba he querido comprobar cuál es la cantidad de nodos del árbol que se evalúan al llamar al algoritmo MiniMax. Para ello he creado una variable global "EVALUADOS" para mantener el contador total de nodos evaluados; esta variable se incrementa en 1 justo antes de la evaluación del nodo (tablero). Luego se imprime por pantalla y se vuelve a inicializar a 0. Esto sucede en cada turno de la máquina.

El código añadido para la prueba viene indicado como "EXPERIMENTO 3".

```python
EVALUADOS = 0 #EXPERIMENTO 3

def busca(tablero, columna):
    i = 0
    while i < tablero.getAlto() and tablero.getCelda(i,columna) == 0:
        i += 1
    i -= 1

    return i

def juega(tablero, posicion):
    global EVALUADOS #EXPERIMENTO 3
    mejorJugada = -100000

    for columna in range(0,tablero.getAncho()):
        fila = busca(tablero,columna)

        if fila != -1:
            nuevoTablero = Tablero(tablero)
            nuevoTablero.setCelda(fila,columna,2)
            valorJugada = V(nuevoTablero,0,False,ALPHA,BETA)

            if valorJugada > mejorJugada:
                mejorJugada = valorJugada
                posicion[0] = fila
                posicion[1] = columna

    print(EVALUADOS) #EXPERIMENTO 3
    EVALUADOS -= EVALUADOS #EXPERIMENTO 3

def V(tablero, profundidad, MAX, alpha, beta):
    global EVALUADOS #EXPERIMENTO 3

    if profundidad == PROFUNDIDAD:
        EVALUADOS += 1 #EXPERIMENTO 3
        return tablero.evaluarJugada()
```



6.4. Primera función de evaluación descartada

En este caso, es un experimento que no llegó a un mejor resultado. Pensé que si en la función de evaluación añadía una comprobación extra a las fichas que están en 3 en fila, podía mejorarla. Esta comprobación era, que si la ficha que estaba en 4 posición consecutiva estaba vacía entonces se contaba, sino no. Al hacerlo comprobé que la inteligencia artificial jugaba peor.

```python
    def horizontal(self, i, j, casilla):
        valor = 0

        if (j+1) < self.getAncho():
            if self.getCelda(i,j+1) == casilla:
                if casilla == 2:
                    valor += 10
                else:
                    valor -= 10

        if (j+2) < self.getAncho():
            if self.getCelda(i,j+1) == casilla and self.getCelda(i,j+2) == casilla and self.getCelda(i,j+3) == 0: #EXPERIMENTO 4
                if casilla == 2:
                    valor += 100
                else:
                    valor -= 100

        if (j+3) < self.getAncho():
            if self.getCelda(i,j+1) == casilla and self.getCelda(i,j+2) == casilla and self.getCelda(i,j+3) == casilla:
                if casilla == 2:
                    valor += 1000
                else:
                    valor -= 1000

        return valor
```

6.5. Segunda función de evaluación descartada

Este experimento se basa en una nueva versión de la función de evaluación.

Se basa en la misma función de evaluación que está implementada en mi algoritmo, pero pensada de una forma diferente, para que no tenga que hacer tantas comprobaciones. La función antigua de evaluación se extiende en muchas más líneas de código.

La idea es que compruebe primero si hay hueco para un jugada en concreto y luego compruebe si hay o no fichas conectadas.

```
def vertical(self, i, j, casilla):
    valor = 0

    if (i+3) < self.getAlto():
        if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla and self.getCelda(i+3,j) == casilla:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1000
        else:
            if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla:
                if casilla == 2:
                    valor += 100
                else:
                    valor -= 100
            else:
                if self.getCelda(i+1,j) == casilla:
                    if casilla == 2:
                        valor += 10
                    else:
                        valor -= 10
    else:
        if (i+2) < self.getAlto():
            if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla:
                if casilla == 2:
                    valor += 100
                else:
                    valor -= 100
            else:
                if self.getCelda(i+1,j) == casilla:
                    if casilla == 2:
                        valor += 10
                    else:
                        valor -= 10
        else:
            if (i+1) < self.getAlto():
                if self.getCelda(i+1,j) == casilla:
                    if casilla == 2:
                        valor += 10
                    else:
                        valor -= 10

    return valor
```

6.6. Tercera función de evaluación descartada

Esta función de evaluación es el primer prototipo creado de la función de evaluación, y estuvo implementada de esta manera hasta que me di cuenta de que debía hacer más comprobaciones.

Se basaba en comprobar si el algoritmo funciona de mejor forma si no se hacen comprobaciones de si la siguiente posición es una posición vacía o no.

```
def vertical(self, i, j, casilla):
    valor = 0

    if (i+1) < self.getAlto():
        if self.getCelda(i+1,j) == casilla:
            if casilla == 2:
                valor += 10
            else:
                valor -= 10

    if (i+2) < self.getAlto():
        if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla:
            if casilla == 2:
                valor += 100
            else:
                valor -= 100

    if (i+3) < self.getAlto():
        if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla and self.getCelda(i+3,j) == casilla:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1000

    return valor
```

6.7. Cuarta función de evaluación descartada

Esta versión de la función de evaluación es similar a las demás presentadas con comprobaciones, pero esta vez también se añade comprobar que la posición siguiente es de nuestro equipo o es del equipo contrario.

```
def vertical(self, i, j, casilla, contraria):
    valor = 0

    if (i+1) < self.getAlto():
        if self.getCelda(i+1,j) == casilla:
            if casilla == 2:
                valor += 10
            else:
                valor -= 10

    try:
        if (i+2) < self.getAlto():
            if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla and (self.getCelda(i+3,j) != contraria or self.getCelda(i+3,j) == 0):
                if casilla == 2:
                    valor += 100
                else:
                    valor -= 100
    except:
        valor += 0

    if (i+3) < self.getAlto():
        if self.getCelda(i+1,j) == casilla and self.getCelda(i+2,j) == casilla and self.getCelda(i+3,j) == casilla:
            if casilla == 2:
                valor += 1000
            else:
                valor -= 1000

    return valor
```

6.8. Máquina contra máquina

```
for x in range(8):
    pygame.init()
    reloj = pygame.time.Clock()
    screen = pygame.display.set_mode([700,620])
    pygame.display.set_caption("Practica 1")
    game_over = False
    tablero = Tablero(None)
    col = -1
    contador = 0
    posicion = [6,x]

    while not game_over:
        contador += 1

        if contador == 1:
            tablero.setCelda(posicion[0],posicion[1],1)

            posicion2 = [-1,-1]
            juega2(tablero,posicion2)
            tablero.setCelda(posicion2[0],posicion2[1],2)

        else:
            posicion1 = [-1,-1]
            juega1(tablero,posicion1)
            tablero.setCelda(posicion1[0],posicion1[1],1)

            if tablero.cuatroEnRaya() == 1:
                game_over = True
                ganaMaquina1 += 1
                print("gana máquina 1")
            else:
                posicion2 = [-1,-1]
                juega2(tablero,posicion2)
                tablero.setCelda(posicion2[0],posicion2[1],2)

                if tablero.cuatroEnRaya() == 2:
                    game_over = True
                    ganaMaquina2 += 1
                    print("gana máquina 2")
```

Este experimento es uno proporcionado por el centro, pero no el código, sino la idea de hacerlo. Se basa en crear un nuevo juego "Conecta4", pero esta vez máquina-máquina.

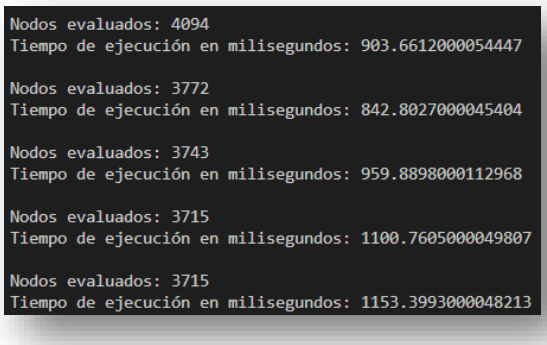
El este fragmento de código del fichero "main", puede observar cómo he conseguido que jueguen entre ellas. No se usan 2 tipos distintos de MiniMax ni de función de evaluación. Se realizan dos bucles de 8 partidas, empezando en cada una de las columnas, en el primer bucle comienza la máquina 1 y en el segundo bucle comienza la máquina 2.

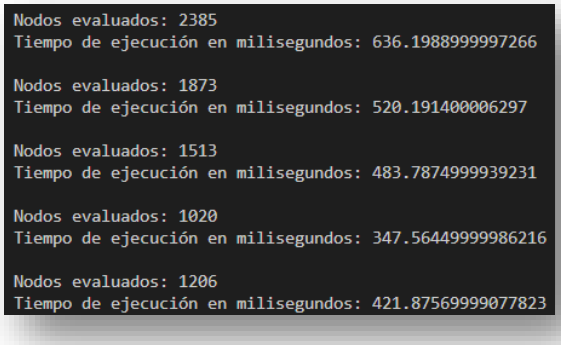
6.9. MiniMax vs Alpha-Beta

Este último experimento consiste en la comparación de rendimiento, nodos evaluados y tiempo de respuesta, entre el algoritmo MiniMax y el algoritmo MiniMax con la poda Alpha-Beta implementada. Vamos a comprobar si es cierto que funciona la poda y que el algoritmo es más eficiente, haciendo que la máquina se comporte de manera más inteligente.

Para comprobarlo, voy a jugar la misma partida, poniendo las fichas exactamente en el mismo lugar, y veremos los resultados de cada uno.

MiniMax vs Alpha-Beta





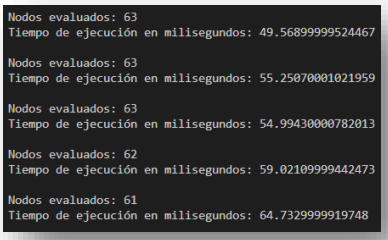
Los dos fragmentos de la ejecución están sacados del inicio de la partida, podemos ver como claramente con la poda Alpha-Beta hace que el algoritmo vaya muchísimo más rápido y consigue que se evalúen una cantidad casi 3 veces menor de nodos/tableros.

7. Comparativa de tiempos y número de nodos generados

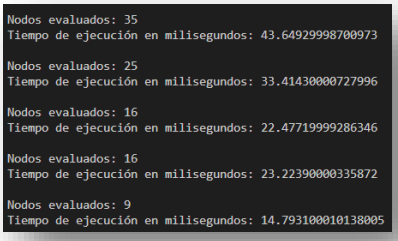
En todas las profundidades se puede apreciar como al principio de la partida, la cantidad de nodos generados es muchísimo mayor que al estar terminando la partida, pues la máquina al inicio tiene más posibilidades donde poner la posición, hay más posiciones vacías y por lo tanto una cantidad mayor de jugadas posibles. Mientras menos posiciones libres queden, menos nodos se generan, y menos tiempo tarda la máquina en decidirse, y por lo tanto podemos jugar más rápido al final de la partida. Los ejemplos usan Alpha-Beta.

7.1. Profundidad 2

Inicio

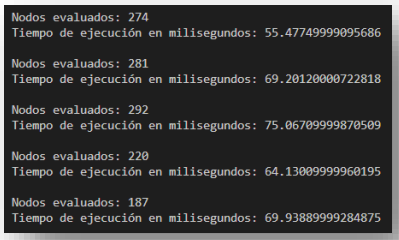


Final

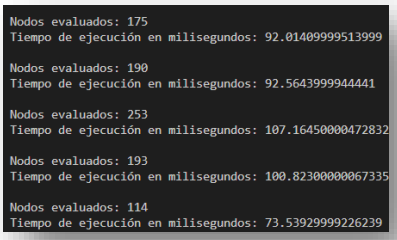


7.2. Profundidad 3

Inicio

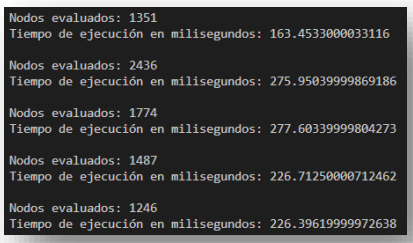


Final

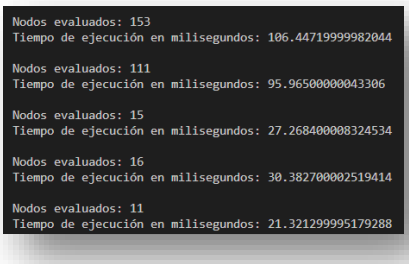


7.3. Profundidad 4

Inicio

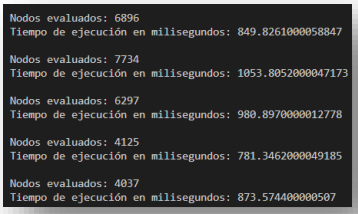


Final

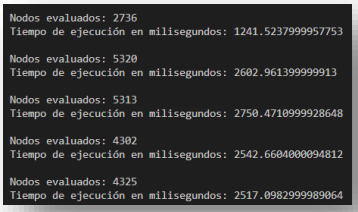


7.4. Profundidad 5

Inicio

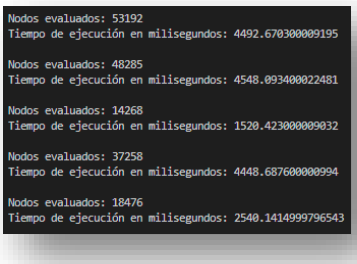


Final

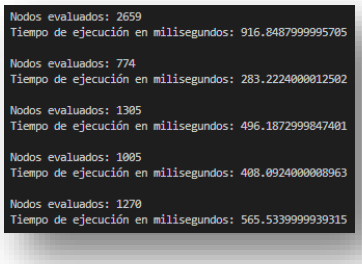


7.5. Profundidad 6

Inicio

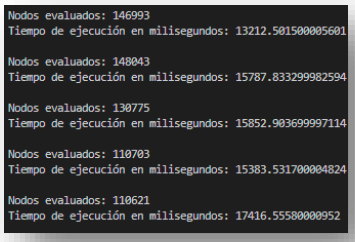


Final

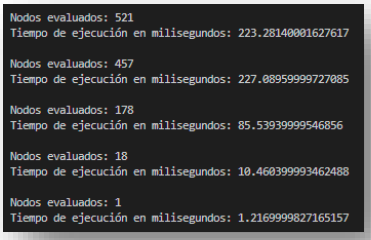


7.1. Profundidad 7

Inicio

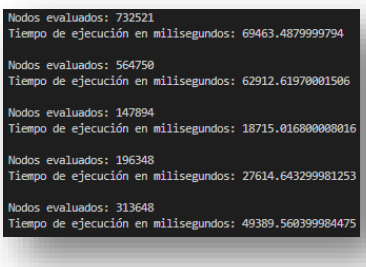


Final

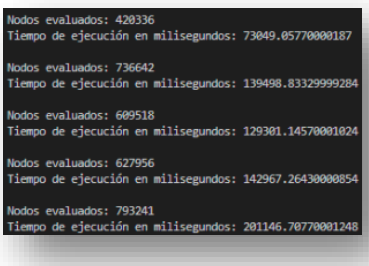


7.1. Profundidad 8

Inicio



Final



8. Conclusiones

Al comenzar con la práctica, lo hice sin conocimiento y me vi en la situación de no saber ni por dónde empezar, por ello antes de empezar a implementar el código me informé sobre las inteligencias artificiales, en cómo se forma el árbol de posibilidades y en el algoritmo MiniMax, consultando foros, videos, artículos etc.

Comencé sin un objetivo claro y pasados los días, hablando con el profesor y con mis compañeros, pude sacar conclusiones y tener una por dónde empezar. Al principio estaba muy frustrado, pues mi algoritmo no iba nada bien. Una vez terminado me siento orgulloso de haber creado una máquina que "piense" por ella misma.

Con respecto al uso de la poda Alpha-Beta, la conclusión que se saco es que es totalmente necesaria su utilización, pues nos permite ahorrar una cantidad abrumadora de nodos/tableros evaluados, permitiendo un uso más eficiente del algoritmo, así como poder jugar de una manera más rápida y contra una máquina que piensa de una manera más eficiente.

La función de evaluación es el elemento principal de la IA, pues el algoritmo MiniMax se puede representar e implementar de diferentes maneras, pero si la función de evaluación es incorrecta o está mal planteada, nada de esto servirá. La inteligencia de la máquina se basa en ella. Para poder hacer una función de evaluación que sea correcta y que se acerque a la perfección, hay que pensar en todos los casos posibles en los que se podría dar una jugada.