2022

27 / 10 / 2022

Practica 1

SISTEMAS INTELIGENTES

ADRIAN UBEDA TOUATI 50771466R

UNIVERSIDAD DE ALICANTE | Grupo 03-3

Contenido

Función de evaluación

Lo primero en lo que he pensado para realizar la función de evaluación ha sido en jugadas de libro, jugadas las cuales el jugador tendría una gran facilidad de ganar si no se cortan al instante. O jugadas que podrían llevar a una victoria segura y no deben ser desaprovechadas.

Estas jugadas se utilizarán durante el proceso de la partida o para intentar finalizarla, pero no para iniciar la partida.

Casos base. Presentación de los mismos y explicación de por qué se han escogido

La función de evaluación tiene en cuenta varios casos los cuales son:

Fin de partida

Este caso es el más sencillo, se limita a detectar si en el tablero hay un 4 en raya, en el caso de ser así, le suma 100000 al resultado si le pertenece el 4 en raya, o le suma -100000 si es del rival

Defensa

Este caso a sido el primero en ser desarrollado.

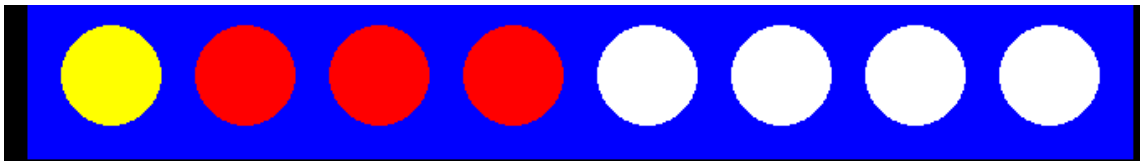
Para realizar la defensa de la máquina, he pensado en ingresar jugadas de libro, puesto que hay jugadas que pueden llevar al jugador ganar en un instante y que serían difícil de detectar contado el número de fichas alineadas.

Las he separado de 2 formas “Defensa critica” y “Defensa casi critica”

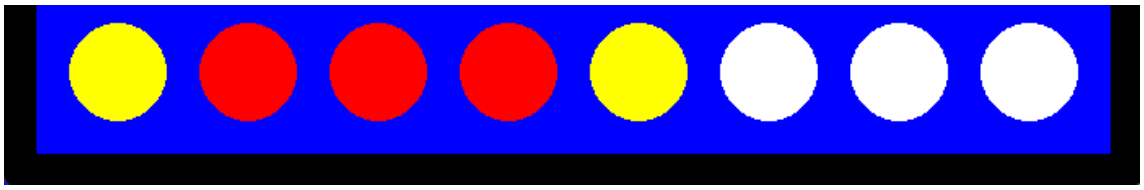
Defensa critica

Defensa critica: corresponde a una jugada la cual el jugador solo tiene que poner una ficha para ganar

Por ejemplo:



En esta jugada la maquina debe cubrir el 3 en raya o sino perderá la partida por lo que responde de la única forma posible

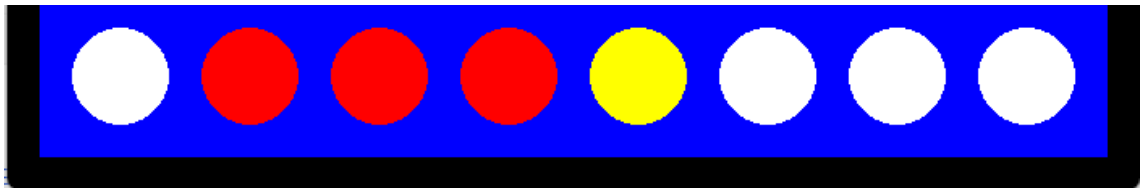


Impidiendo así perder

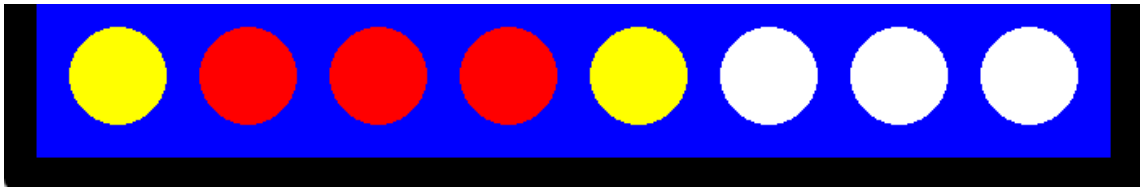
Estas jugadas se le corresponden el valor de -1000 puntos, puesto que deben evitarse a toda costa ya que ponen en jaque la partida

Las otras jugadas de libro en defensa critica son:

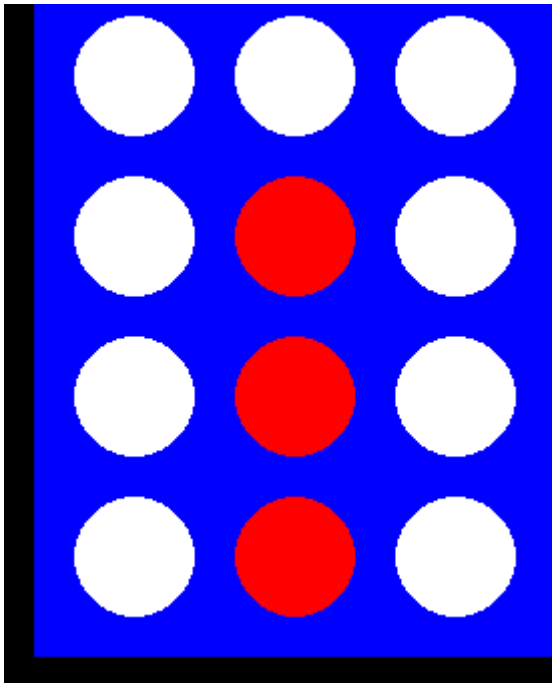
Caso planteado 1



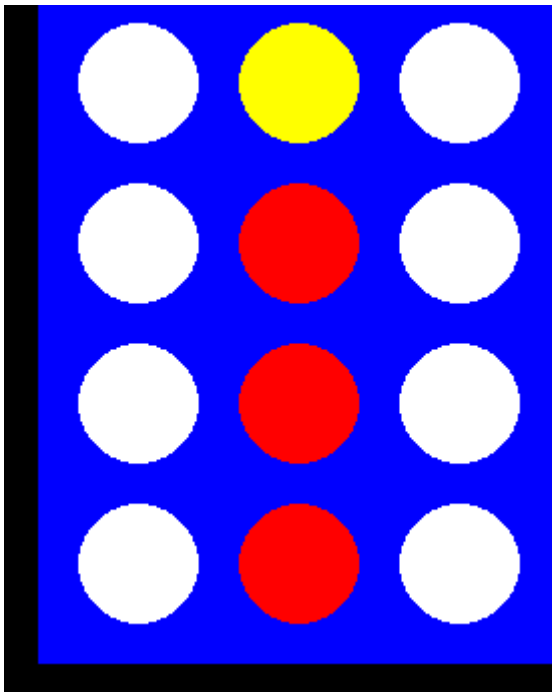
Respuesta 1



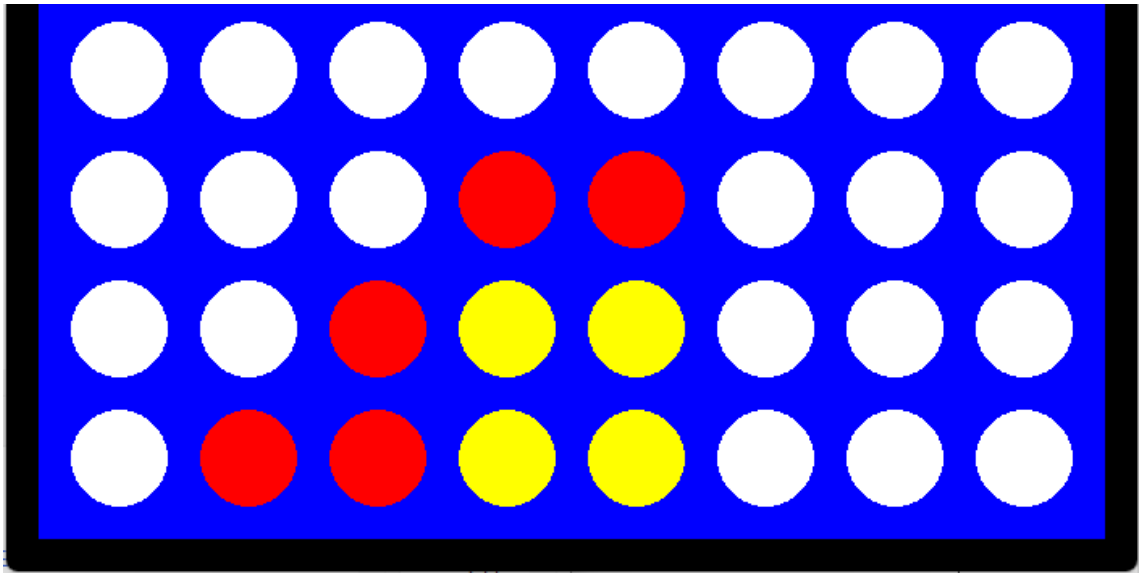
Caso planteado 2



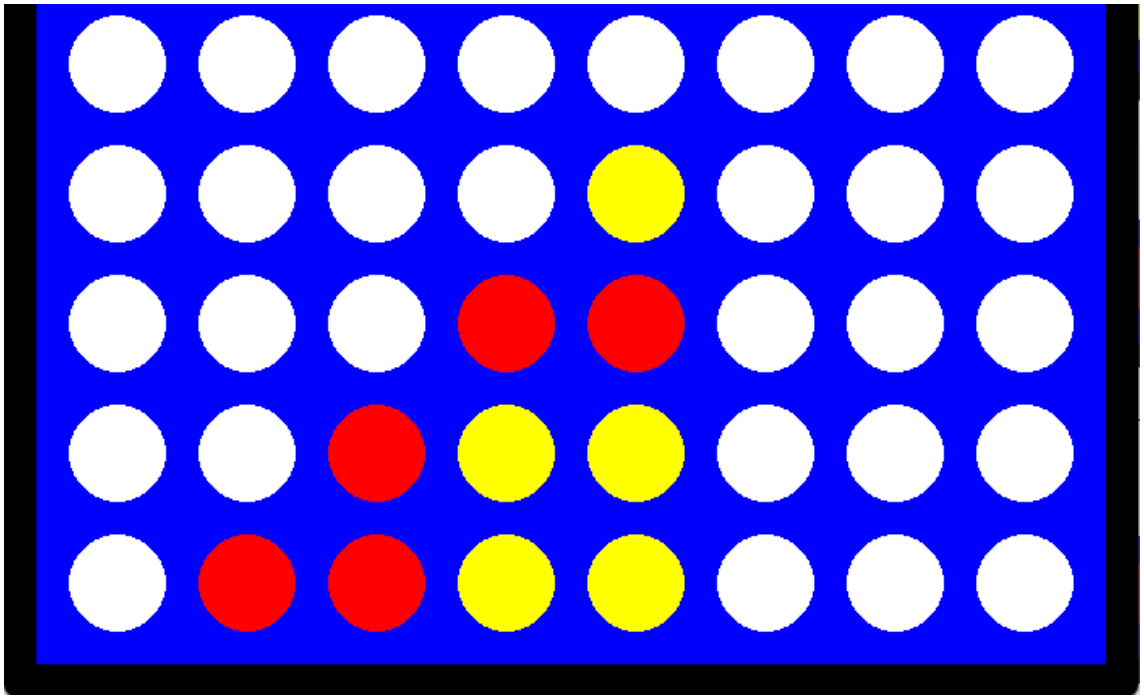
Respuesta 2



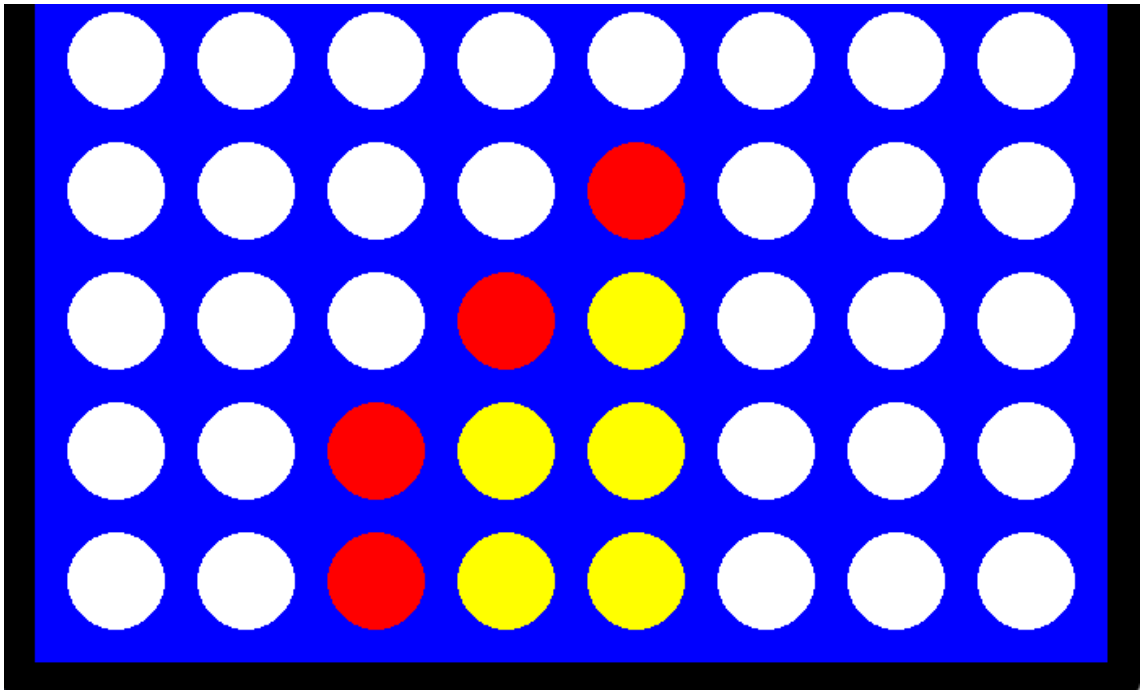
Caso planteado 3



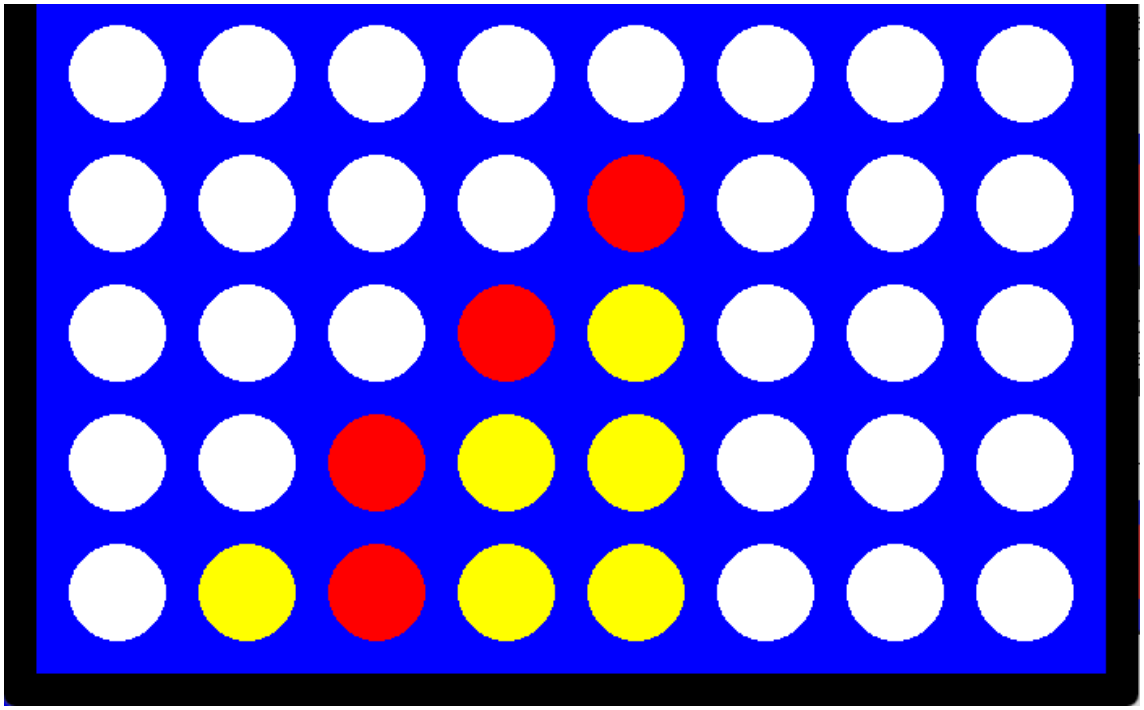
Respuesta 3



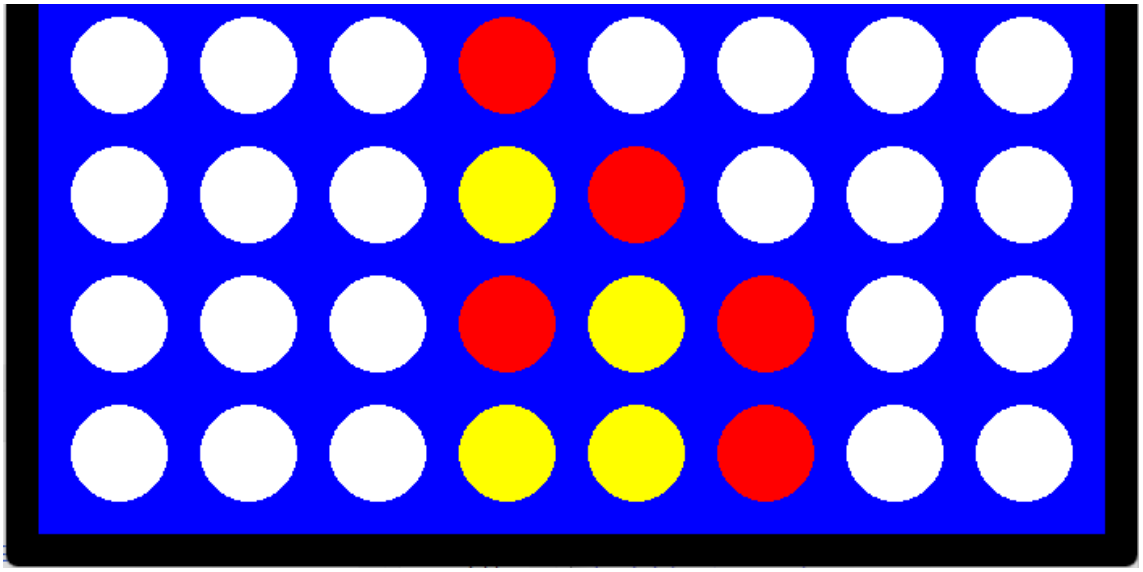
Caso planteado 4



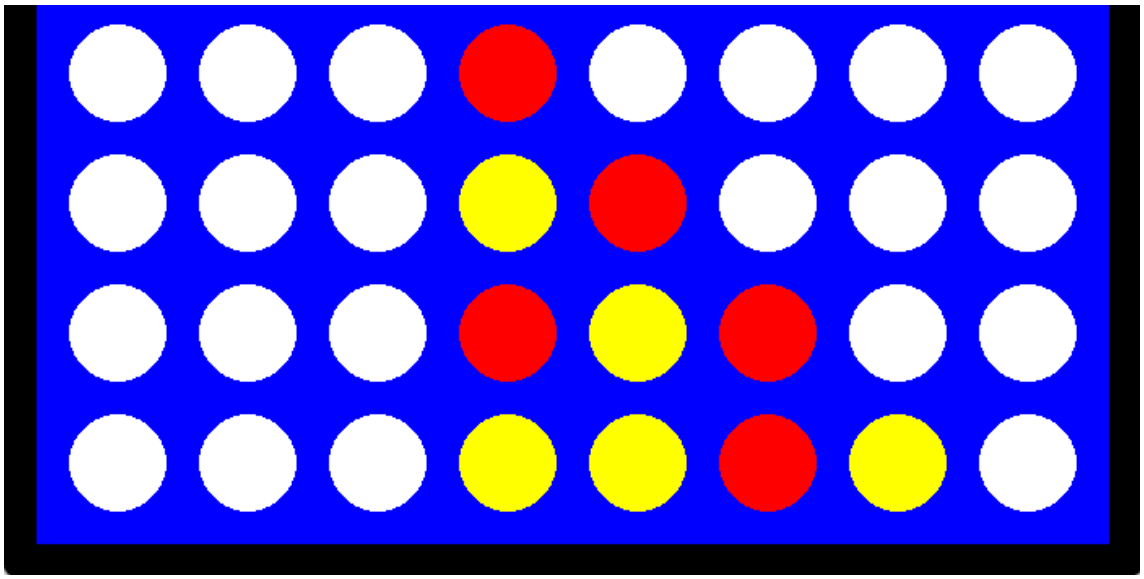
Respuesta 4



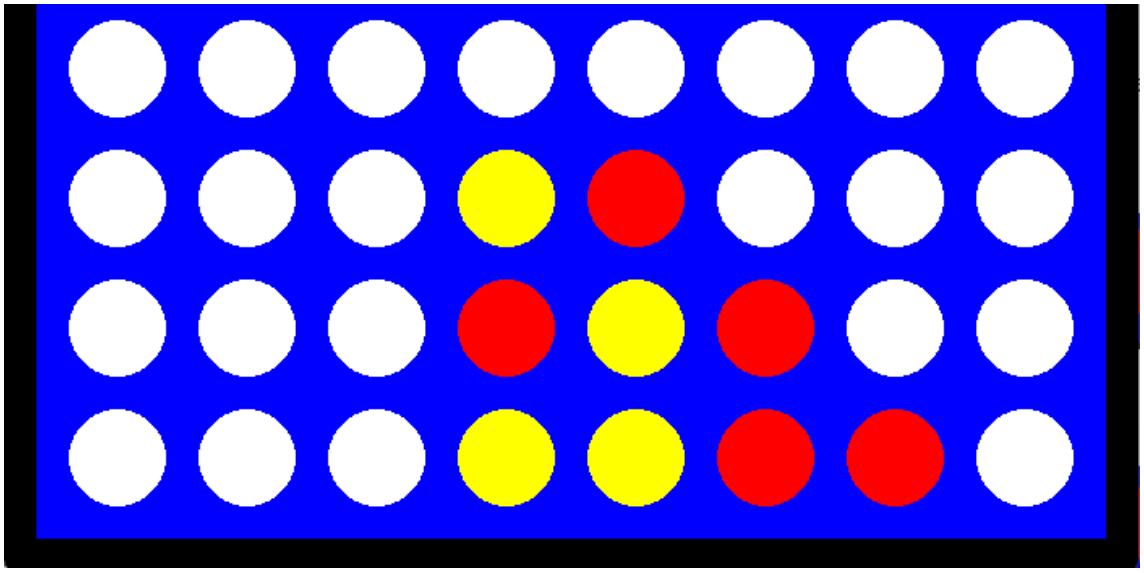
Caso planteado 5



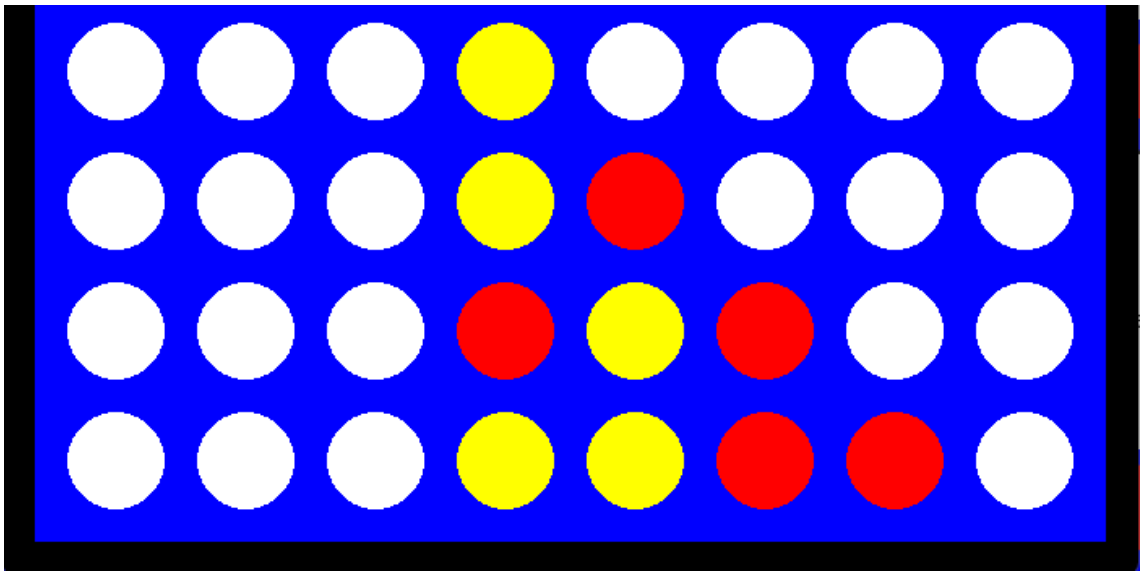
Respuesta 5



Caso planteado 6



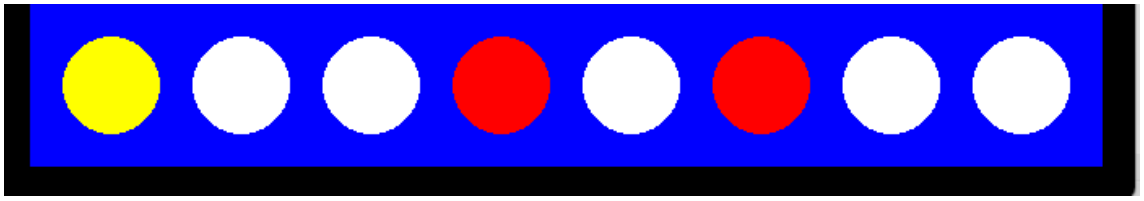
Respuesta 6



Defensa semi critica

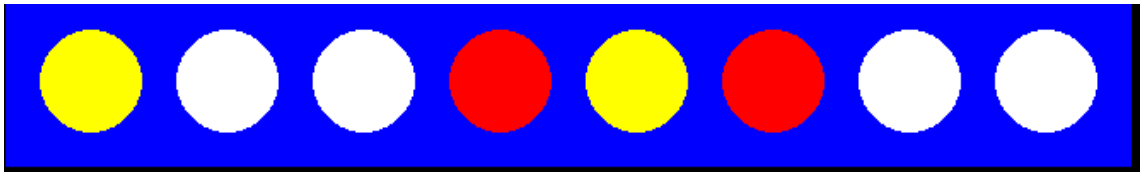
Defensa semi critica: corresponde a una jugada la cual el jugador solo tiene que poner una ficha mas para garantizar la victoria en la siguiente jugada

Por ejemplo:



Esta jugada no se puede detectar usando un escaneo de pantalla y contando el número de fichas que hay alineadas. Por lo que si la maquina no pone su ficha entre las 2 rojas, habrá perdido.

Este caso está considerado en la función de evaluación, por lo que la maquina reaccionara de esta forma:

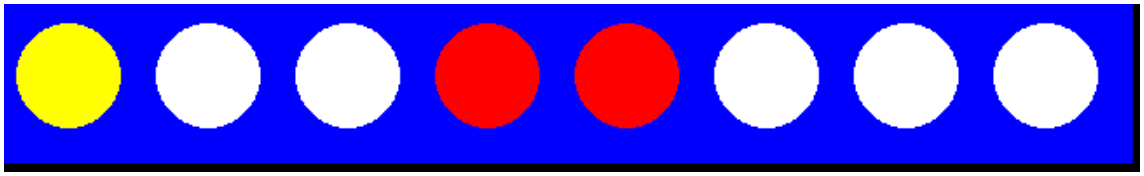


Impidiendo así perder.

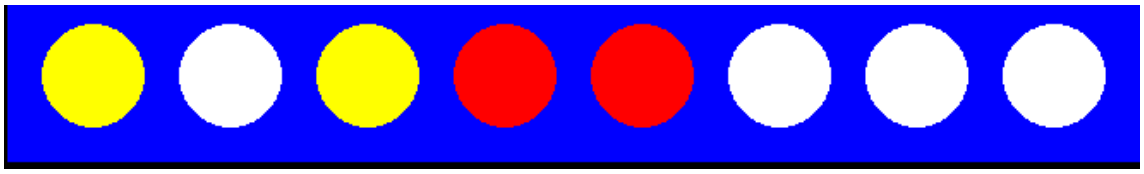
Estas jugadas se le corresponden el valor de -100 puntos, puesto que deben evitarse, aun siendo graves no lo son tanto como las criticas puesto que hay 2 jugadas de margen, por ello se le asigna menos puntuación

Las otras jugadas de libro en defensa semi critica son:

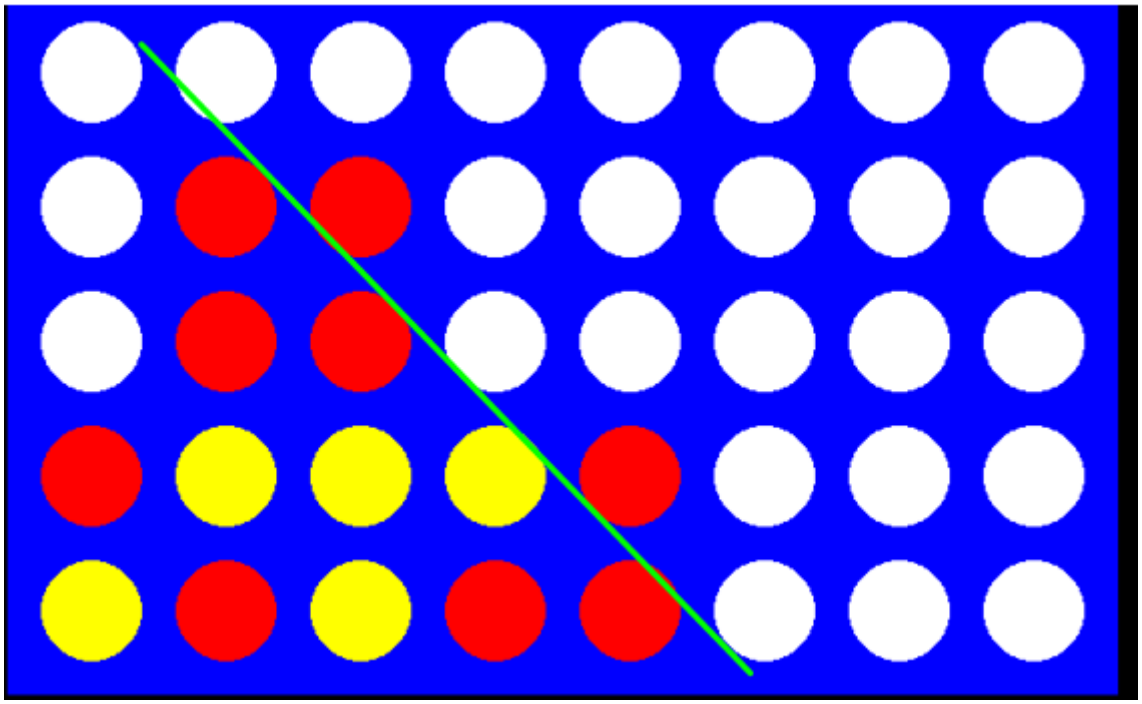
Caso planteado 1



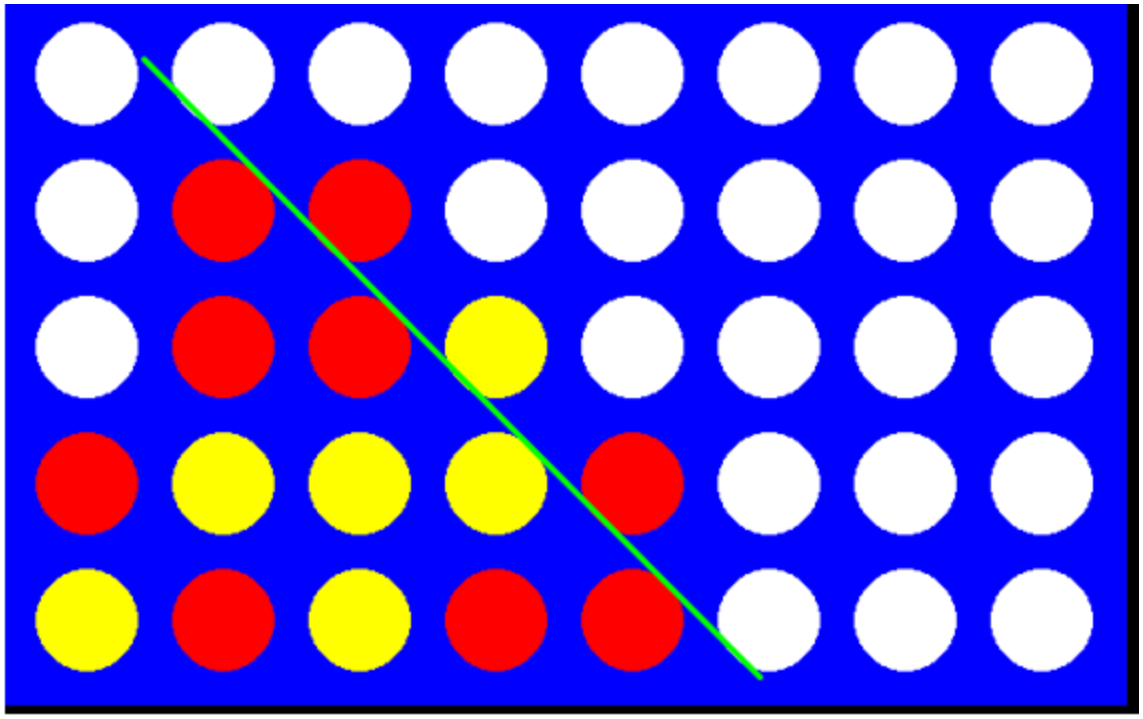
Respuesta 1



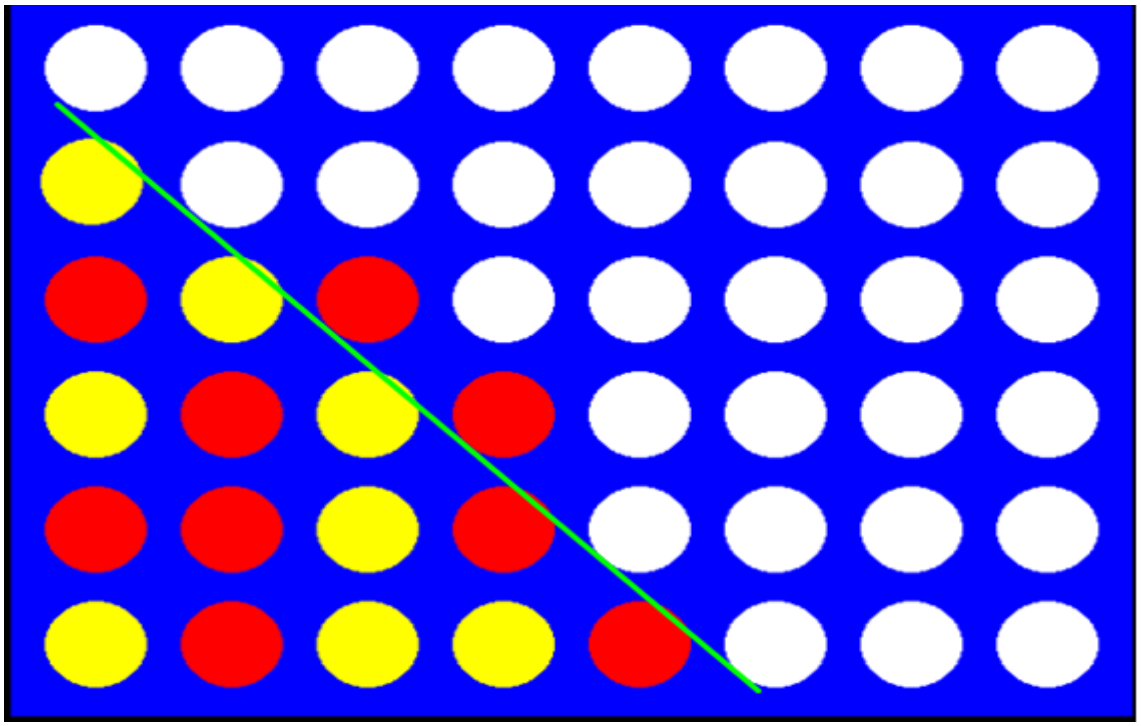
Caso planteado 2



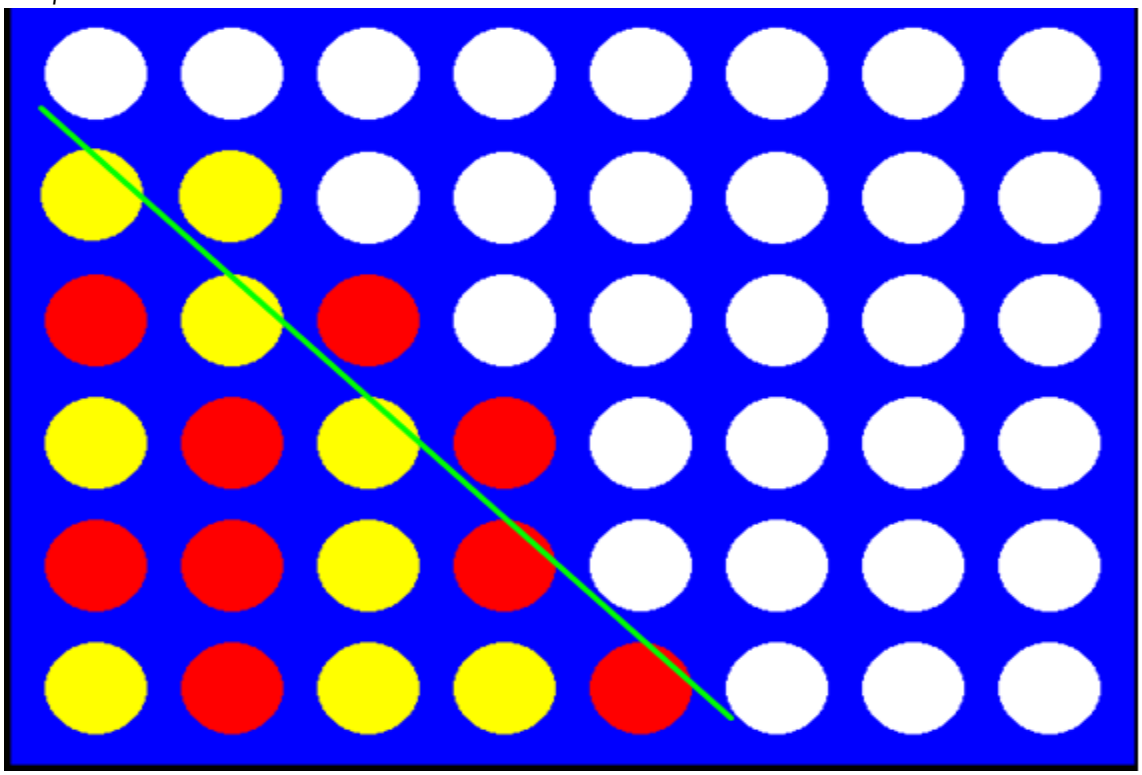
Respuesta 2



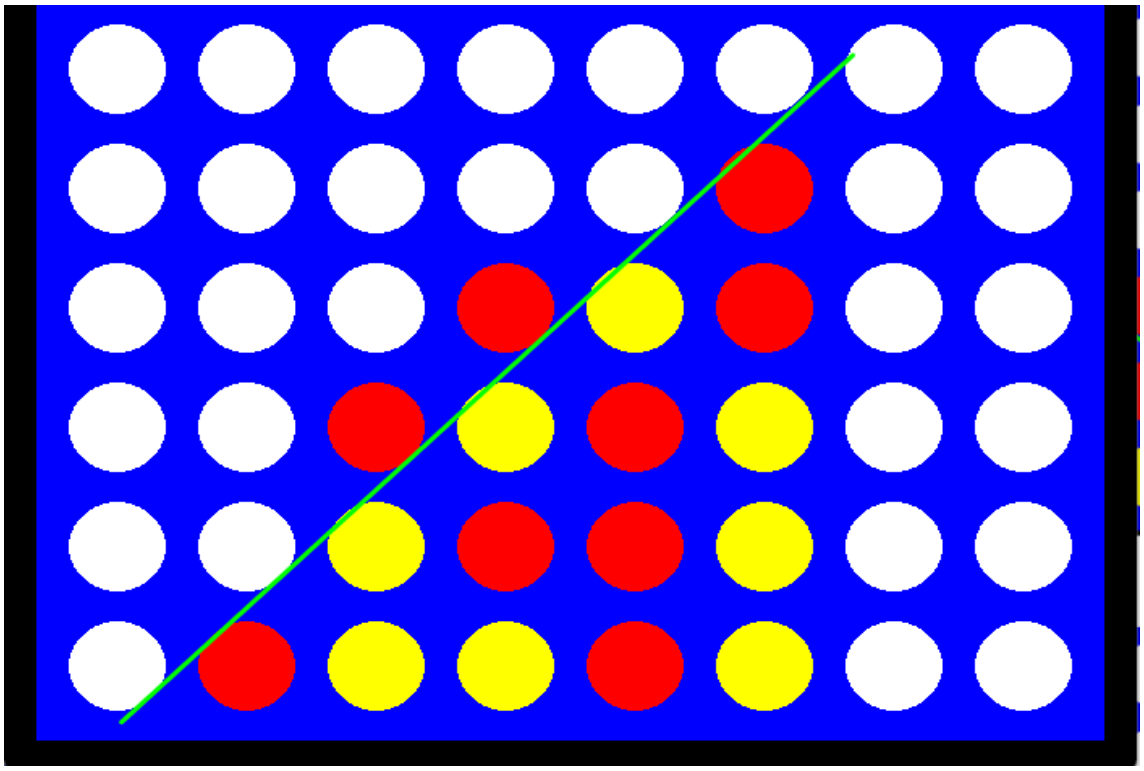
Caso planteado 3



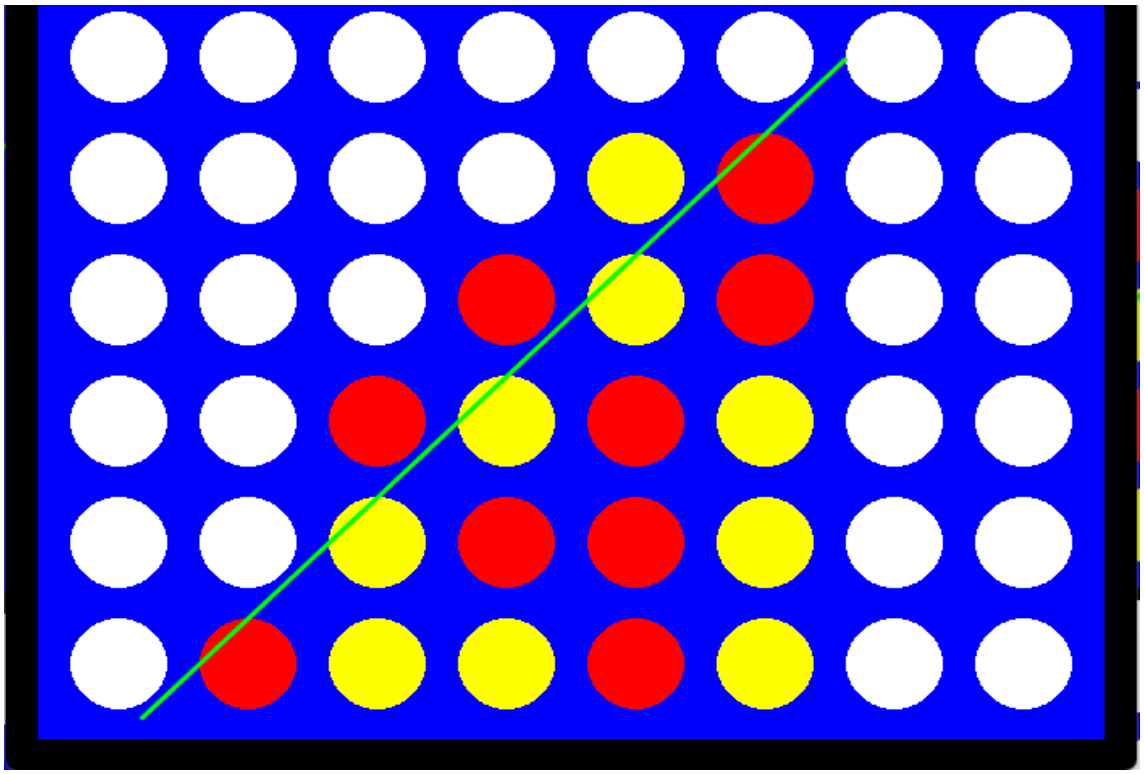
Respuesta 3



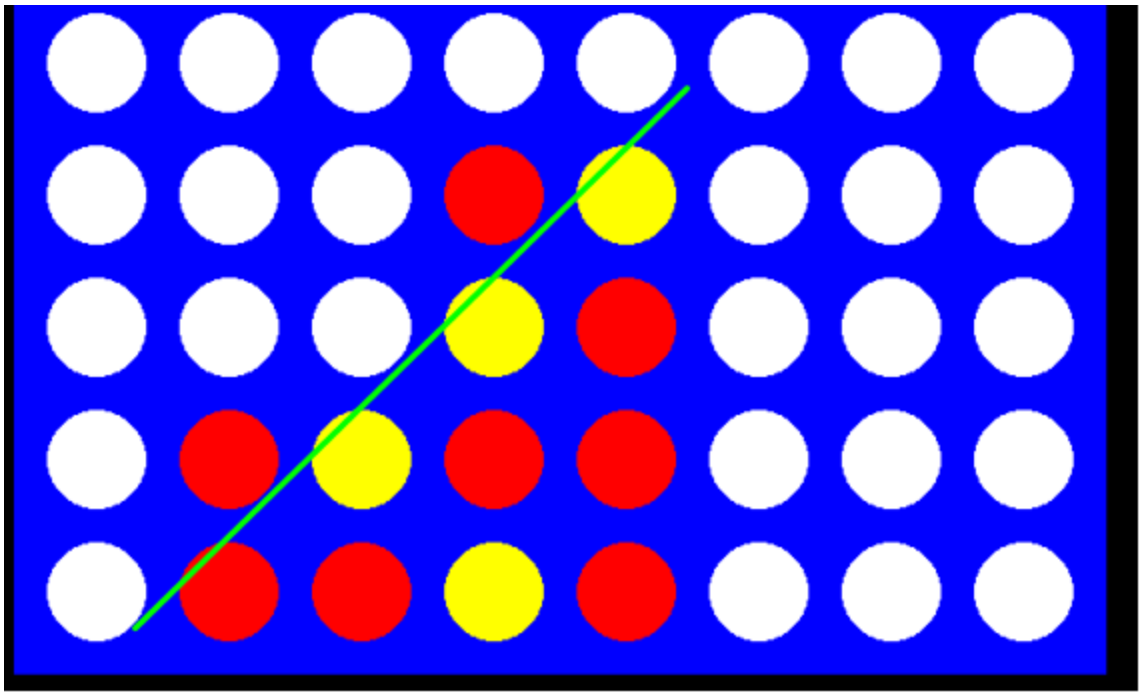
Caso planteado 4



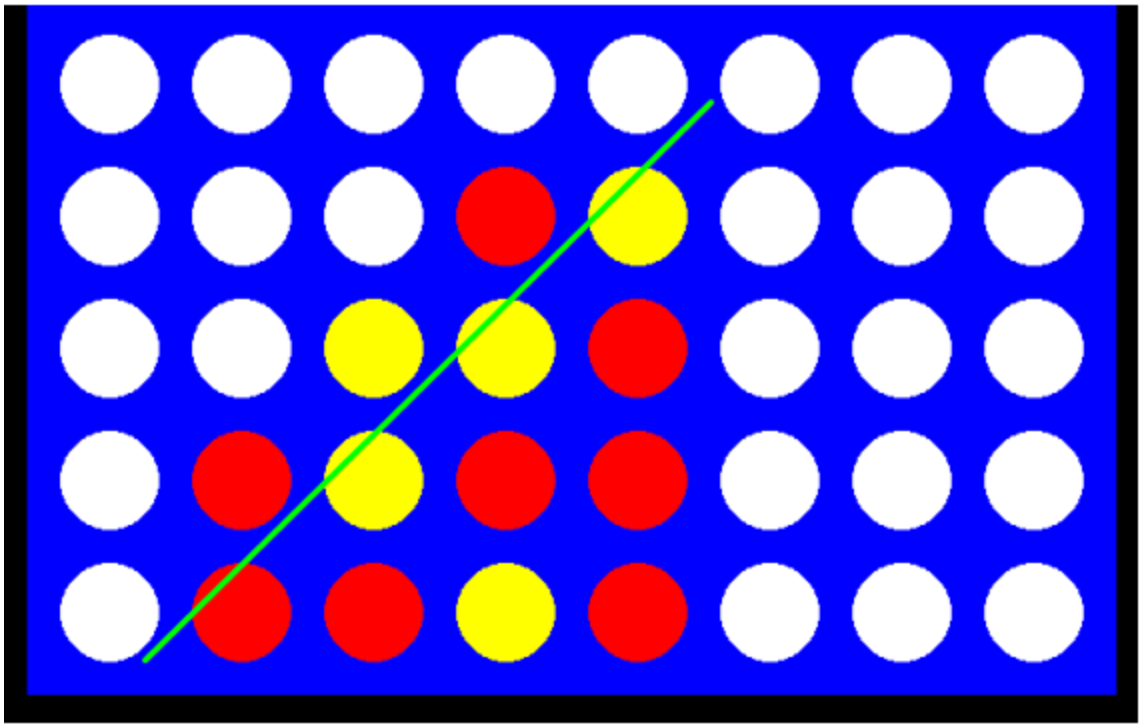
Respuesta 4



Caso planteado 5



Respuesta 5



Ataque

El ataque fue implementado después de la defensa, implemente la función de detección de la defensa de tal forma que cambiando un booleano, alterna los jugadores, por lo que la función de defensa se vuelve la de ataque, puesto que son los mismos casos.

Cabe destacar que la puntuación es igual que la defensa, pero en el caso de ataque critico vale 1000 y ataque casi crítico 100

Caso intermedio

El caso intermedio: Todo caso que no corresponde a una jugada critica o casi crítica, es decir la que generara esos casos.

Cabe destacar que la mayor parte de la partida corresponde a este caso.

Para poder implementarla he evaluado cada jugada intermedia de 2 formas, "Posible 4 en raya" y "Matriz"

Posible 4 en raya

Esta función tiene el rol de poder visualizar cuantos 4 en raya son posible formarse para cada ficha, además de eso también tiene en cuenta otros factores como si en el supuesto caso de que se formase ese 4 en raya las fichas tendrían un suelo, y también verifica si hay fichas ya puestas para formarlo, por lo que incita a poner fichas en raya.

```
casillaTieneSuelo = 1

posibleCuatroEnRaya = 2

fichaEnRaya = 5
```

Suma y resta a la par según el tablero

He probado 2 funciones de evaluaciones, una con esta funcionalidad y otra sin, el resultado ha sido que con la matriz el algoritmo es más eficiente y gana más partidas.

Matriz

Para complementar el posible 4 en raya, se me ocurrió hacer una matriz del tablero, para así intentar tomar las posiciones claves y generar más fácilmente 4 en raya.

En un principio me diseñe la matriz pensando sobre todo en el principio de la partida, para que así pudiese tomar la delantera, pero después me di cuenta

de que esta matriz hacía que a la larga el jugador tomase el centro, mientras que la maquina se enfocaba en los lados, lo cual le daba desventaja.

```
   0 1 2 3 4 5 6 7
0  1 2 3 4 4 3 2 1
1  2 3 4 5 5 4 3 2
2  3 4 5 6 6 5 4 3
3  4 5 6 7 7 6 5 4
4  5 6 7 8 8 7 6 5
5  6 7 8 9 9 8 7 6
6  7 8 9 10109 8 7
```

Por ello se me ocurrió una forma de obtener la mejor matriz posible, la idea era de contar todos los posibles 4 en raya que se pueden generar en una casilla y atribuirle ese valor.

```
   0 1 2 3 4 5 6 7
0  3 4 5 7 7 5 4 3
1  4 6 8 10108 6 4
2  5 8 111313118 5
3  7 1013161613107
4  5 8 111313118 5
5  4 6 8 10108 6 4
6  3 4 5 7 7 5 4 3
```

Desde que reemplace la matriz, ganar a la maquina se ha vuelto todo un reto.

Para testear la máquina, le dije a mi hermano que si conseguía ganar le daría 5 euros, el resultado fue maquina: 50 hermano: 0

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

En mi caso no ha sido necesario, puesto que, con las clases puestas a disposición, no he tenido la necesidad de crear una nueva. Analizo los tableros en la propia clase de algoritmos.

Análisis de casos

Analicemos los resultados devueltos por la función de evaluación, para ello simularemos una partida.

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

2
```

Debería dar 0, pero debido a un fallo en la función Posible 4 en taya, devuelve 2, he pensado en quitarla o no, pero la forma de jugar con la matriz únicamente es muy siempre y esta función aporta bastante a máquina, le suma la mayoría de las veces puntuación extra a máquina, y de esta forma es más agresiva.

Al final conseguí arreglar esta función, pero el resultado es que juega peor de esa forma. Por lo que dejare la antigua función defectuosa.

El cambio realmente es mínimo, puesto que se le suma 2 siempre.

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

5
```

Gracias a la matriz, la función evaluación cree que la maquina tiene ventaja, puesto que esa zona de la matriz tiene más puntos, en cuestión 3 más que el jugador, pero al sumarle 2 por la función defectuosa, se queda en 5

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

11
```

Misma situación que antes

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

14
```

Sin cambios relevantes

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

1017
```

La maquina a detectado un posible 4 en raya y por ello se le suma 1000

Saltamos un par de jugadas…

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

-985
```

Y aquí el resultado le resta -1000 porque detecta un posible 4 en raya enemigo

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

25
```

Que es cortado rápidamente

Avanzamos un poco la partida

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

101035
```

Y aquí se le suma 100000 porque hay un 4 en raya de la maquina

Y así ganaría la maquina

Queda comprobar los casos casi críticos

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

-98
```

-100 + 2 del error de la función

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

102
```

100 + 2 del error de la función

Y con esto terminaríamos el análisis de los resultados de la función de evaluación

Gracias a este apartado me he dado cuenta del error de la función 4 en Raya, el resto de las funciones que se llaman en la función de evaluación están bien equilibradas.

Por temas de rendimiento, he decidido no llamar a la función 4 en Raya y simplemente sumarle +2 a la máquina

Análisis temporal

El algoritmo evalúa es el siguiente:

```
def evalua(tablero,coordenadas):
    resultado = 0

    cuatroEnRayaAtaque = 100000
    cuatroEnRayaDefensa = -100000

    ganador = tablero.cuatroEnRaya()
    if ganador != 0 and ganador == 1:
        resultado += cuatroEnRayaDefensa
    elif ganador == 2:
        resultado += cuatroEnRayaAtaque

    jugadaFinalAtaque = 1000
    jugadaFinalDefensa = -1000

    jugadaPeligrosaAtaque = 100
    jugadaPeligrosaDefensa = -100


    resultado += jugadaFinalAtaque *JugadaFinal(tablero,True,coordenadas)
    resultado += jugadaFinalDefensa * JugadaFinal(tablero,False,coordenadas)
    resultado += jugadaPeligrosaAtaque * JugadaPeligrosa(tablero,True,coordenadas)
    resultado += jugadaPeligrosaDefensa * JugadaPeligrosa(tablero,False,coordenadas)

    #Jugadas intermedias
    resultado += 2#PosibleCuatroEnRaya(tablero,coordenadas)

    resultado += Intermedio(tablero,coordenadas)

    return resultado
```

Independientemente de la entrada de la función, las estancias con el cuadrado en rojo se ejecutarán siempre el mismo número de veces, por lo que podemos contar todo el conjunto de cuadrados rojos como 1 paso.

Sin embargo, los cuadrados verdes, dependen de una función, por lo que tenemos que analizar primero esas funciones para poder determinar la complejidad de la función evalúa

Función tablero.cuatroEnRaya()

```
def cuatroEnRaya(self):
    i=0
    fin=False
    ganador=0

    while not fin and i<self.getAlto():
        j=0
        while not fin and j<self.getAncho():
            casilla=self.getCelda(i,j)
            if casilla!=0:
                #búsqueda en horizontal
                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:
                        ganador=casilla
                        fin=True
                #búsqueda en vertical
                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:
                        ganador=casilla
                        fin=True
                #búsqueda en diagonal
                if (i+3) <self.getAlto():
                    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:
                            ganador=casilla
                            fin=True
                    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:
                            ganador=casilla
                            fin=True
            j=j+1
        i=i+1
    return ganador
```

Podemos ver que le pasamos un tablero de entrada a la función, los cuadrados en rojo se ejecutaran una vez, ya que no dependen de la variable de entrada.

El cuadrado en verde, sin embargo, depende de la altura del tablero pasado como parámetro.

El cuadrado en azul depende igualmente del parámetro de entrada, pero al mismo tiempo esta anidado al verde.

Calculemos el peor caso:

$$1 + \sum_{i=0\ldots alto-1} (1 + \sum_{j=0\ldots ancho-1} 1)$$

$$1 + \sum_{i=0\ldots alto-1} (1 + (ancho - 1 - 0) + 1)$$

$$1 + \sum_{i=0\ldots alto-1} (ancho + 1)$$

$$1 + (alto) * (ancho + 1)$$

$$1 + alto + alto * ancho$$

$$\mathrm{O}\ (\text{alto} * \text{ancho})$$

El mejor caso seria en el que el tablero fuese de altura 0 por lo que no entraría en el bucle y seria constante

$$\Omega\ (1)$$

Función JugadaFinal(tablero,True/False,coordenadas)

Parte de la función:

```
def JugadaFinal(tablero,ataque,coordenadas):
    jugAnalizado = 0
    if ataque == True:
        jugAnalizado = 2
    else:
        jugAnalizado = 1

    puntuacion = 0

    i = -1
    j = -1

    for coordenada in coordenadas:
        i = coordenada[0]
        j = coordenada[1]

        #"-->."
        if j < 5:
            if tablero.getCelda(i,j) == jugAnalizado:
                if tablero.getCelda(i,j+1) == 0 and tablero.getCelda(i,j+2) == jugAnalizado and tablero.getCelda(i,j+3) == jugAnalizado:
                    if SueloFicha(tablero,(i,j+1)):
                        puntuacion +=1
```

Podemos ver que le pasamos un tablero de entrada a la función, los cuadrados en rojo se ejecutaran una vez, ya que no dependen de la variable de entrada.

El cuadrado en verde, sin embargo, depende del número de coordenadas.

Calculemos el peor caso:

$$1 + \sum_{i=0 \ldots coordenadas} 1$$

$$1 + coordenadas$$

$$\mathrm{O}\,(coordenadas)$$

El mejor caso sería que no haya coordenadas, y entonces que no entre en el bucle

$$\Omega\,(1)$$

Función JugadaPeligrosa (tablero,True/False,coordenadas)

Esta función es idéntica a la anterior en términos de complejidad temporal

Parte de la función:

```
def JugadaPeligrosa(tablero,ataque,coordenadas):
    jugAnalizado = 0
    if ataque == True:
        jugAnalizado = 2
    else:
        jugAnalizado = 1

    i = -1
    j = -1

    puntuacion = 0

    for coordenada in coordenadas:
        i = coordenada[0]
        j = coordenada[1]

        #"_._._"
        if j < 4:
            if tablero.getCelda(i,j) == 0:
                if tablero.getCelda(i,j+1) == jugAnalizado and tablero.getCelda(i,j+2) == 0 and tablero.getCelda(i,j+3) == jugAnalizado and tablero.getCelda(i,j+4) == 0:
                    if SueloFicha(tablero,(i,j+2)) and SueloFicha(tablero,(i,j)) and SueloFicha(tablero,(i,j+4)):
                        puntuacion += 1
```

Podemos ver que le pasamos un tablero de entrada a la función, los cuadrados en rojo se ejecutaran una vez, ya que no dependen de la variable de entrada.

El cuadrado en verde, sin embargo, depende del número de coordenadas.

Calculemos el peor caso:

$$1 + \sum_{i=0 \ldots coordenadas} 1$$

$$1 + coordenadas$$

$$\mathrm{O}\,(coordenadas)$$

El mejor caso sería que no haya coordenadas, y entonces que no entre en el bucle

$$\Omega\,(1)$$

Función Intermedio (tablero ,coordenadas)

Esta función es idéntica a la anterior en términos de complejidad temporal

Parte de la función:

```
def Intermedio(tablero,coordenadas):
    puntuacion = 0
    global tableroConteo
    for cordenada in coordenadas:
        if tablero.getCelda(cordenada[0],cordenada[1]) == 1:
            puntuacion -= tableroConteo.getCelda(cordenada[0],cordenada[1])
        if tablero.getCelda(cordenada[0],cordenada[1]) == 2:
            puntuacion += tableroConteo.getCelda(cordenada[0],cordenada[1])

    return puntuacion
```

Podemos ver que le pasamos un tablero de entrada a la función, los cuadrados en rojo se ejecutaran una vez, ya que no dependen de la variable de entrada.

El cuadrado en verde, sin embargo, depende del número de coordenadas.

Calculemos el peor caso:

$$1 + \sum_{i=0 \ldots coordenadas} 1$$

$$1 + coordenadas$$

$$\mathrm{O}\,(coordenadas)$$

El mejor caso sería que no haya coordenadas, y entonces que no entre en el bucle

$$\Omega\,(1)$$

Minimax

¿Cómo se ha implementado un nodo?

Hay 2 tipos de nodos, los nodos terminales y el resto de los nodos.

Nodos terminales

Los nodos terminales son aquellos que no tendrán hijos.

El resultado será pasado a los nodos intermedios, consiste en una tupla con 3 parámetros:

- La puntuación resultante de la función de evaluación.
- La coordenada de la última ficha ingresada
- La profundidad en la cual se ha alcanzado el nodo terminal

Hemos identificado 3 tipos de nodos terminales (o casos base):

```
if n == 0 or tablero.cuatroEnRaya() or TableroLleno(tablero):
    posicionesObjetivo = CoordenadasImportantes(tablero)
    return (evalua(tablero,posicionesObjetivo),coordenada,aux)
```

- ***n==0*:** Este caso base, se refiere a cuando la profundidad pasada como parámetro ha sido alcanzada, por lo que estos serán nodos hijos.
- ***tablero.cuatroEnRaya():*** Si en el tablero existe algún 4 en raya, la partida termina, por lo que no hay que generar mas hijos, y esto lo transforma en un nodo terminal
- ***TableroLleno(tablero):*** Verifica si el tablero esta lleno, en este caso, no se podrían poner mas fichas, por lo que no hay que generar mas hijos, y lo vuelve un caso terminal

Nodo intermedio

Son aquellos que manejan la recursión, reciben un resultado de la misma función, se separa en 2 bloques, “Mini” y “Max”

Max

```
else:
    if turno:
        for j in range(tablero.ancho):

            i = busca(tablero,j)

            if i == -1:
                continue

            tableroAux = Tablero(tablero)
            tableroAux.setCelda(i,j,2)

            coordenada = (i,j)

            resultado = minimax(tableroAux,n-1,False,coordenada,aux+1)
            resultados.append((resultado[0],coordenada,resultado[2]))

        resultado = resultados[0]
        for puntos in resultados:
            if puntos[0] > resultado[0]:
                if puntos[0] > 80000 and resultado[0] > 80000:
                    if resultado[2] > puntos[2]:
                        resultado = puntos
                else:
                    resultado = puntos
            elif puntos[0] == resultado[0] and resultado[2] > puntos[2]:
                resultado = puntos
            elif puntos[0] == resultado[0] and resultado[2] == puntos[2]:
                aleatorio = random.randint(0, 1)
                if aleatorio == 0:
                    resultado = puntos

        return resultado
```

El primer bloque es el encargado de realizar la recursividad, mientras que el segundo tiene el objetivo de seleccionar el resultado entre todos los obtenidos.

Hablemos del primer bloque, empezamos buscando la altura de la columna en la que estamos, si esta esta llena se cortara esa iteración del bucle. En el caso de que no este llena, crea un tablero auxiliar, copia del original, y le añade una ficha de la máquina, después guardamos la coordenada y realizamos la recursividad.

Como parámetros tiene:

- El tablero auxiliar con la nueva ficha
- La profundidad -1, puesto que ya hemos bajado un nodo más
- Pasamos el booleano a False, puesto que tendrá que saltar al bloque "Mini", si no es un nodo terminal
- Pasamos la coordenada de la nueva ficha
- Aumentamos la variable aux +1, sirve para controlar la profundidad en la que nos situamos

Una vez el resultado recursivo recogido de un caso base u otro intermedio, añadimos en una lista dicho resultado, modificando la coordenada a la del padre, puesto que sino la ficha, no tendría en cuenta el tablero actual de la partida y podría ser colocada en sitios imposibles.

Hablemos ahora del segundo bloque, primero recogemos el primer resultado de la lista para tener un primer valor de salida, seguidamente para cada resultado de la lista, verificamos si este es mas grande que el primer resultado que tenemos, si es el caso lo reescribimos, después verificamos si ha llegado a un 4 en raya, esto se verifica viendo si el resultado es > 80000 y por ultimo vemos si la profundidad de dicho valor es mas pequeña que la anterior, así podrá seleccionar le jugada que este mas cerca de dicho 4 en raya.

En el caso de que la puntuación del resultado sea igual al objeto de la función, verificamos si la profundidad es menor, si es el caso la cambiamos.

Mini

```
else:
    for j in range(tablero.ancho):

        i = busca(tablero,j)

        if i == -1:
            continue

        tableroAux = Tablero(tablero)
        tableroAux.setCelda(i,j,1)

        coordenada = (i,j)

        resultado = minimax(tableroAux,n-1,True,coordenada,aux+1)
        resultados.append((resultado[0],coordenada,resultado[2]))
    resultado = resultados[0]

    for puntos in resultados:
        if puntos[0] < resultado[0]:
            resultado = puntos
        elif puntos[0] == resultado[0] and resultado[2] > puntos[2]:
                resultado = puntos

    return resultado
```

El primer bloque es casi el mismo que en anterior, el único cambio es que a la hora de realizar la llamada recursiva cambiamos el False a True, y así que la siguiente llamada salte al bloque "Max" si no es un nodo terminal

El segundo bloque en cambio varia un poco puesto que es mas simple que el anterior, simplemente verifica que el nuevo valor es mas pequeño que el anterior y en el caso de que sea igual, cual tiene la menor profundidad

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

Devuelve la puntuación, la coordenada de la primera ficha de la jugada objetivo, y un parámetro auxiliar utilizado en la función para controlar la profundidad.

Accedemos a la última jugada sustituyendo la coordenada del hijo por la del padre, y así al final tendremos la primera coordenada de la jugada objetivo, después esa coordenada será recogida en la tupla pasada como resultado.

¿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.

Veamos los tiempos que realiza Minimax con diferentes profundidades

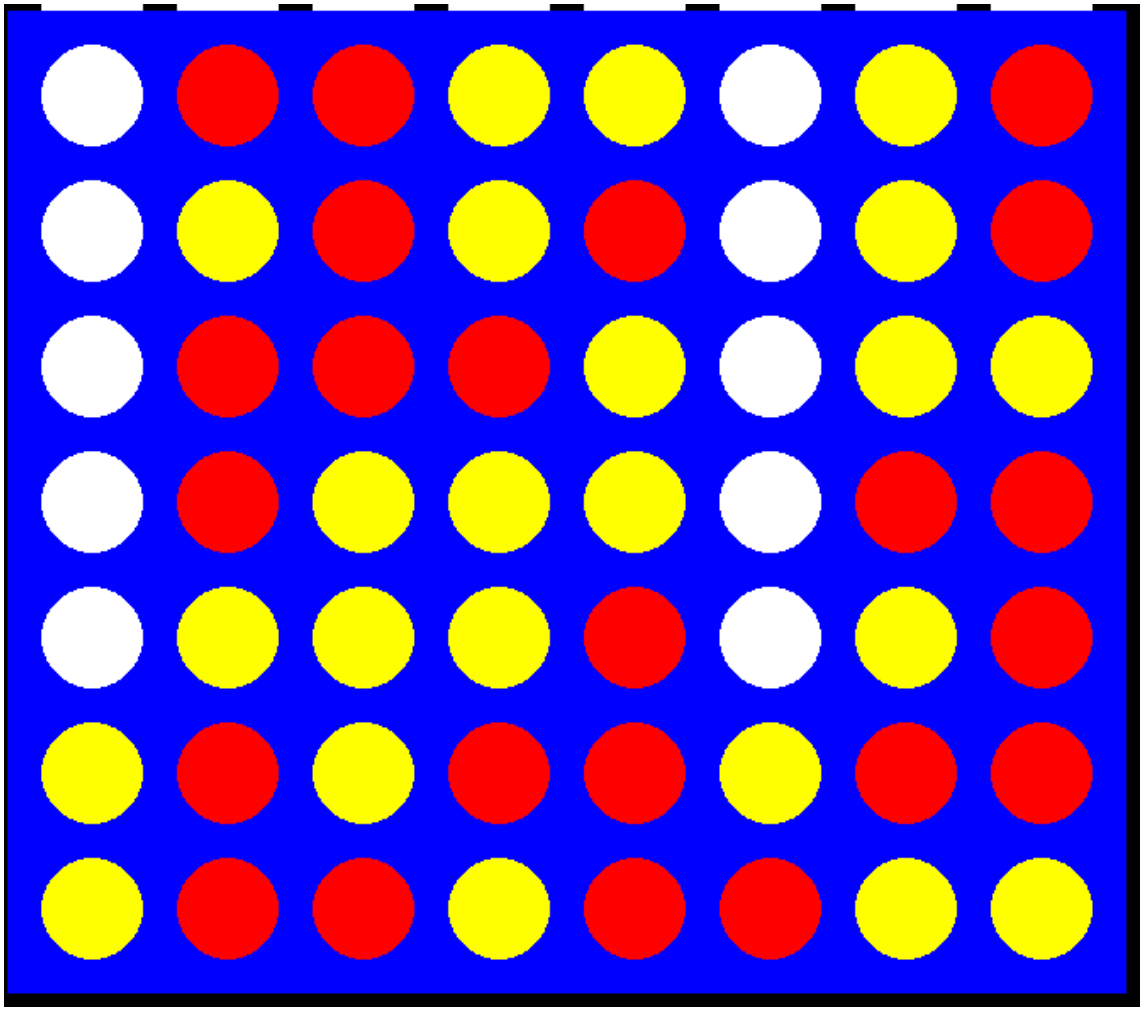
Minimax	tiempo en s
5	0,612406
6	5,211324
7	43,677336
8	351,334873

Por lo que, para jugar cómodamente, tendríamos que jugar con profundidad 5.

¿Cómo juega la máquina en ese nivel? Pon ejemplos

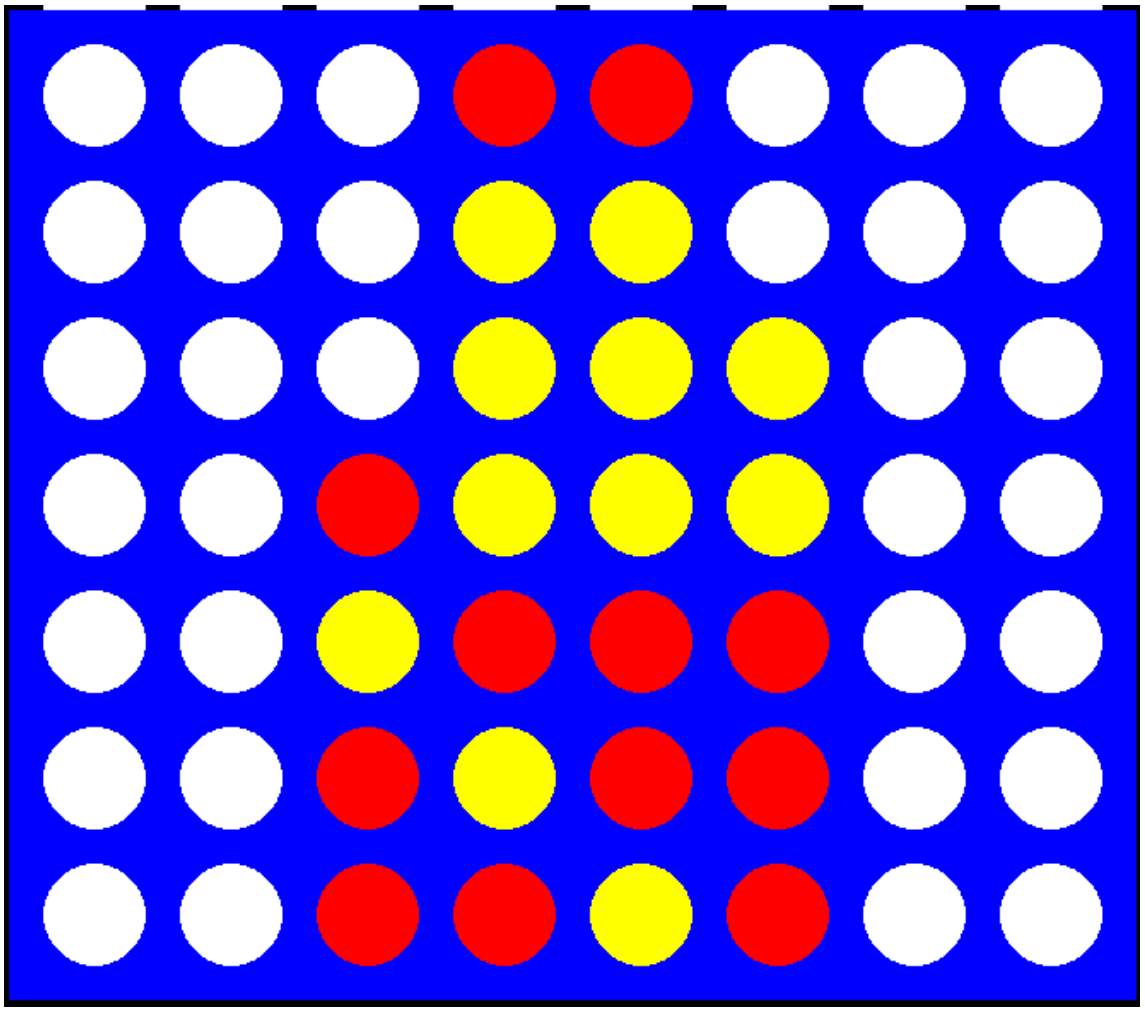
La máquina es capaz de defenderse y de atacar de forma muy efectiva, sobre todo a mitad fin de partida. Al dominar el centro del tablero, tiene estas características

Ejemplo:



Pero al principio de partida, es muy vulnerable si el rival amontona fichas por la parte inferior del tablero.

Ejemplo:



Para poder ser objetivos con el nivel de la máquina, he preguntado a familiares y amigos si podrían jugar 5 partidas para ver los resultados.

	Gana maquina	Pierde maquina	Descripcion del sujeto
Daniel	5	0	Compañero de carrera que cursa SI
Adrian	4	1	Creador
Ana	4	1	Cursa ultimo año del doble grado de ingenieria informatica y matematicas
Niko	5	0	Alumno de 4 de primaria
Eduardo	4	1	Compañero de carrera que cursa SI
Francisco	50	0	Estudiante de primero de bachiller
Total	72	3	

Como podemos ver la maquina gano un total de 72 veces y perdió 3 veces.

Esta media se puede ver inflada por el número de intentos de Francisco, por lo que lo reduciremos a 5, y el número de partidas totales a 30

Calculemos el número de victorias:

$$72 - 45 = 27$$

Por lo que el porcentaje de victorias es del:

$$\frac{27}{30} * 100 = 90\%$$

¿Minimax siempre hace la misma jugada para un determinado tablero? ¿Cómo podríamos evitar esto?

Es un algoritmo el cual, si no le ponemos aleatoriedad, realizara siempre el mismo resultado a una misma jugada, por lo que se puede repetir una misma partida todo el rato, no varia el comportamiento de MiniMax

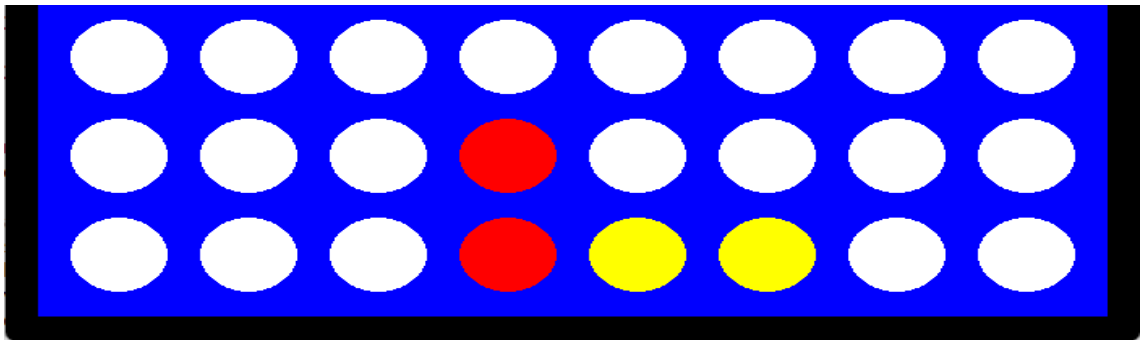
Por lo que para evitar esto, debemos poner un componente aleatorio en la función, que no comprometa su funcionalidad

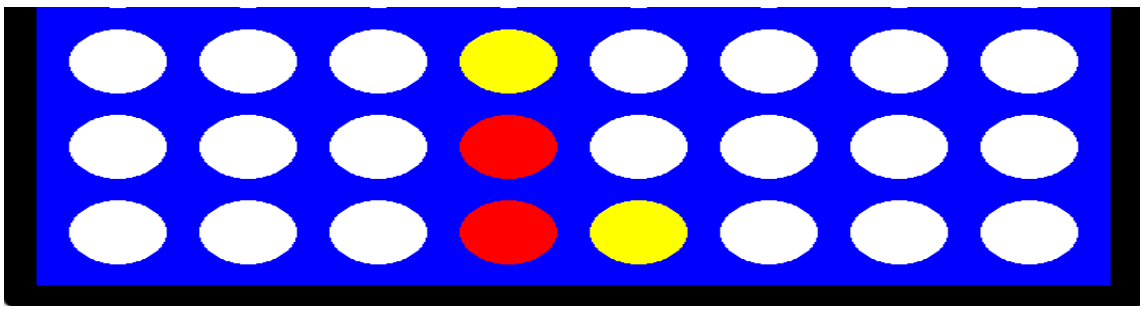
[Opcional] Implementa que minimax (alfa-beta) no realice siempre la misma jugada para nodos con F(N) iguales

En el caso de que tengan tanto la misma puntuación como profundidad, elegiremos cuál de los 2 resultados quedarnos, a través de una función random. Para así evitar que todas las partidas con las mismas jugadas sean iguales

```
elif puntos[0] == resultado[0] and resultado[2] == puntos[2]:
    aleatorio = random.randint(0, 1)
    if aleatorio == 0:
        resultado = puntos
```

Podemos observar 2 jugadas iguales con respuestas diferentes de la maquina





Alfabeta

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

La poda se realiza en la llamada recursiva, primero se actualiza alfa en el caso de "Max" o beta en el caso de "Mini", después se verifica si se realiza la poda, esto ocurre cuando alfa >= beta

```
resultado = alfabeta(tableroAux,n-1,False,coordenada,aux+1,alfa,beta)
if resultado != None:
    resultados.append((resultado[0],coordenada,resultado[2]))
    if resultado[0] > alfa:
        alfa = resultado[0]
    if alfa >= beta:
        break
```

```
resultado = alfabeta(tableroAux,n-1,True,coordenada,aux+1,alfa,beta)
if resultado != None:
    resultados.append((resultado[0],coordenada,resultado[2]))
    if resultado[0] < beta:
        beta = resultado[0]
    if alfa >= beta:
        break
```

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

Alfabeta	tiempo en s
5	0,179622
6	1,116668
7	4,11406
8	20,637101

Considero que un tiempo razonable es aquel que ronde el segundo, por lo que podría alcanzar la profundidad 6

PRACTICA 1

¿Cuántos nodos te ahorras al aplicar alfa-beta vs minimax?

Minimax	numero de nodos
5	8776
6	70216
7	561736
8	4413149

Alfabeta	numero de nodos	nodos ahorrados
5	2394	6382
6	14650	55566
7	51451	510285
8	234320	4178829

Como podemos ver se ahorran una cantidad ingente de nodos

¿Es más rápido que minimax? Muestra ejemplos

Minimax	tiempo en s
5	0,612406
6	5,211324
7	43,677336
8	351,334873

Alfabeta	tiempo en s
5	0,179622
6	1,116668
7	4,11406
8	20,637101

Es mucho más rápido que minimax

Documentación

Se debe dejar claro el objetivo de las pruebas, qué información se ha recopilado a partir de las mismas y qué conclusiones se han obtenido.

El progreso de la practica ha sido lento, las pruebas consistían en simplemente jugar contra la maquina y ver sus defectos y mejorar posteriormente la función de evaluación.

Una gran mejora, fue el cambio de matriz, como mencionamos anteriormente:

En un principio me diseñe la matriz pensando sobre todo en el principio de la partida, para que así pudiese tomar la delantera, pero después me di cuenta de que esta matriz hacía que a la larga el jugador tomase el centro, mientras que la maquina se enfocaba en los lados, lo cual le daba desventaja.

```
   0 1 2 3 4 5 6 7
0  1 2 3 4 4 3 2 1
1  2 3 4 5 5 4 3 2
2  3 4 5 6 6 5 4 3
3  4 5 6 7 7 6 5 4
4  5 6 7 8 8 7 6 5
5  6 7 8 9 9 8 7 6
6  7 8 9 10109 8 7
```

Por ello se me ocurrió una forma de obtener la mejor matriz posible, la idea era de contar todos los posibles 4 en raya que se pueden generar en una casilla y atribuirle ese valor.

```
   0 1 2 3 4 5 6 7
0  3 4 5 7 7 5 4 3
1  4 6 8 10108 6 4
2  5 8 111313118 5
3  7 1013161613107
4  5 8 111313118 5
5  4 6 8 10108 6 4
6  3 4 5 7 7 5 4 3
```

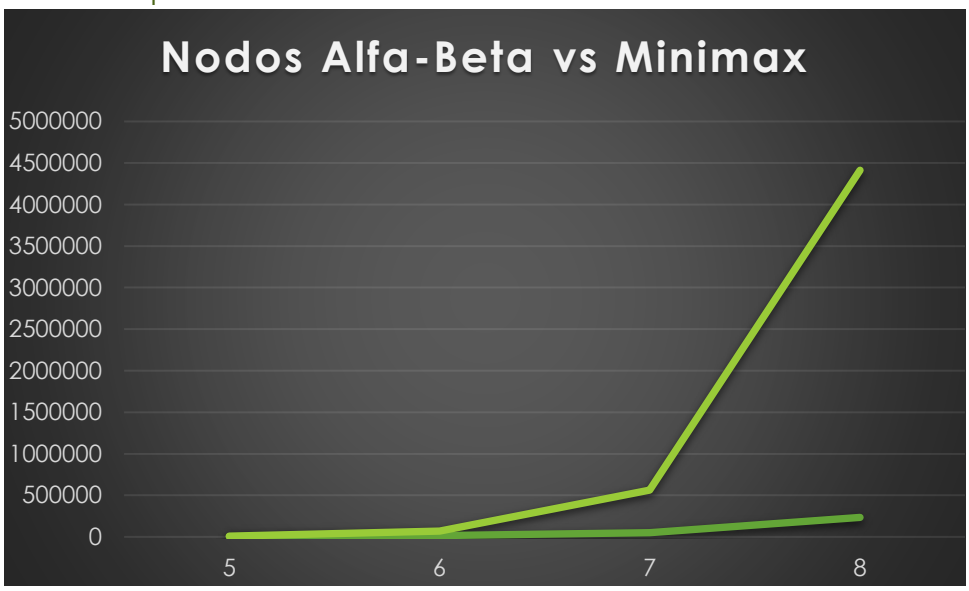
Desde que reemplace la matriz, ganar a la maquina se ha vuelto todo un reto.

Para testear la máquina, le dije a mi hermano que si conseguía ganar le daría 5 euros, el resultado fue maquina: 50 hermano: 0

PRACTICA 1

Haz un análisis comparativo de minimax vs alphabeta para varios niveles de profundidad. Debes mostrar los resultados gráficamente.

Nodos explorados



Minimax	numero de nodos
5	8776
6	70216
7	561736
8	4413149

Alfabeta	numero de nodos
5	2394
6	14650
7	51451
8	234320

Tiempo requerido

Minimax	tiempo en s
5	0,612406
6	5,211324
7	43,677336
8	351,334873

Alfabeta	tiempo en s
5	0,179622
6	1,116668
7	4,11406
8	20,637101

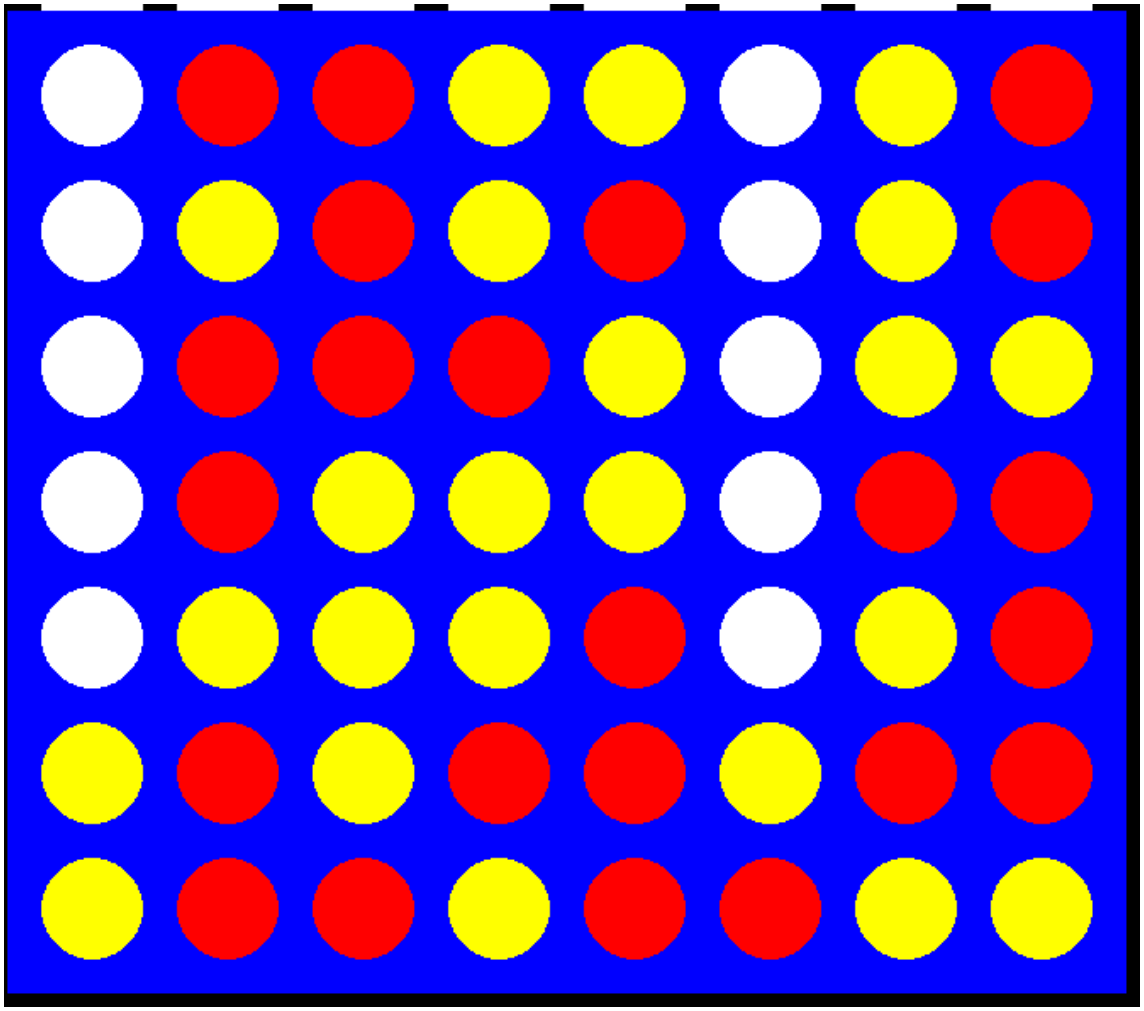
Los dos grafos son casi los mismos puesto que el tiempo se traduce en nodos

Analiza cómo juega tu mejor función de evaluación al detalle. Comenta sus puntos fuertes y débiles

Como dijimos anteriormente en minimax:

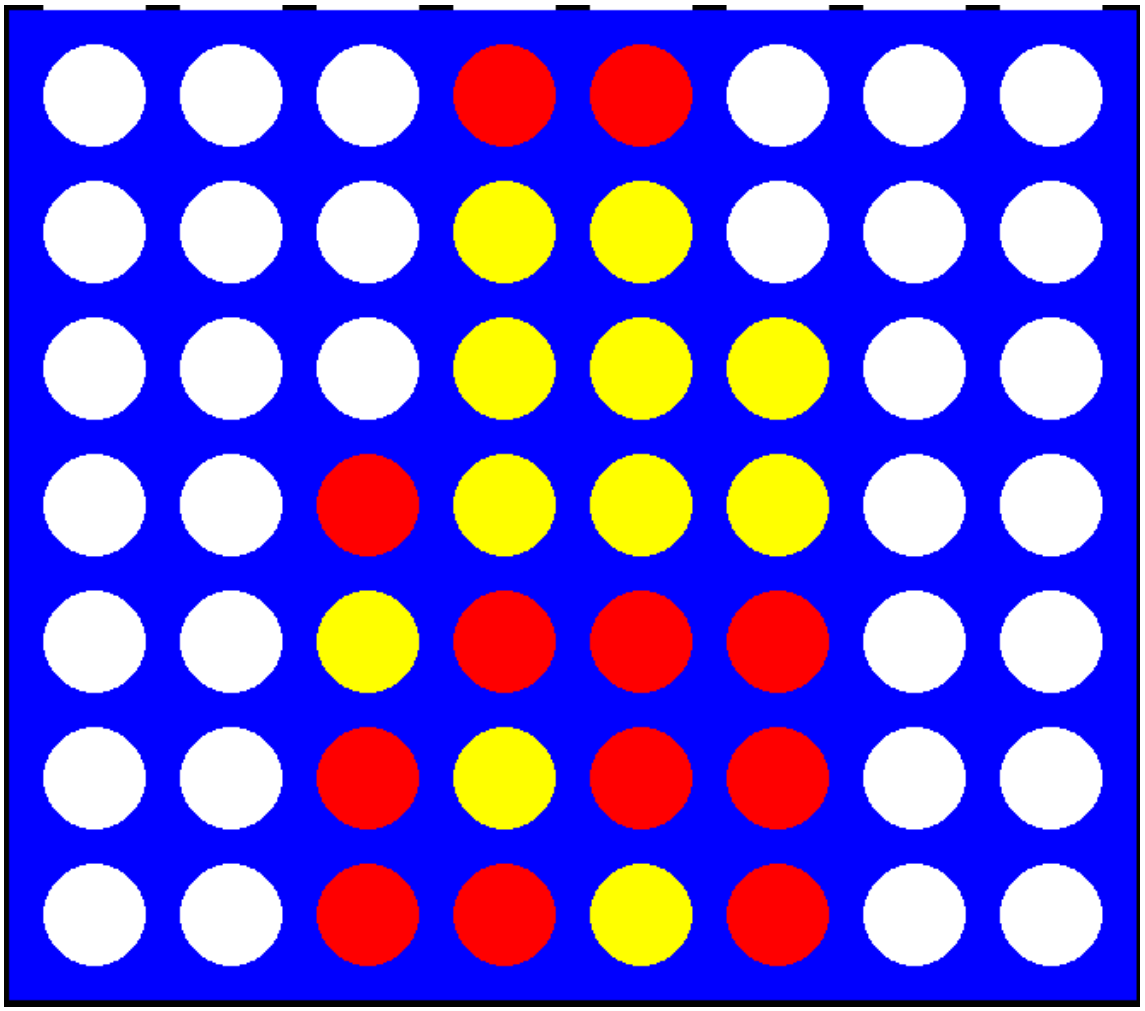
La máquina es capaz de defenderse y de atacar de forma muy efectiva, sobre todo a mitad fin de partida. Al dominar el centro del tablero, tiene estas características

Ejemplo:



Pero al principio de partida, es muy vulnerable si el rival amontona fichas por la parte inferior del tablero.

Ejemplo:



Para poder ser objetivos con el nivel de la máquina, he preguntado a familiares y amigos si podrían jugar 5 partidas para ver los resultados.

	Gana maquina	Pierde maquina	Descripcion del sujeto
Daniel	5	0	Compañero de carrera que cursa SI
Adrian	4	1	Creador
Ana	4	1	Cursa ultimo año del doble grado de ingenieria informatica y matematicas
Niko	5	0	Alumno de 4 de primaria
Eduardo	4	1	Compañero de carrera que cursa SI
Francisco	50	0	Estudiante de primero de bachiller
Total	72	3	

Como podemos ver la maquina gano un total de 72 veces y perdió 3 veces.

Esta media se puede ver inflada por el número de intentos de Francisco, por lo que lo reduciremos a 5, y el número de partidas totales a 30

Calculemos el número de victorias:

$$72 - 45 = 27$$

Por lo que el porcentaje de victorias es del:

$$\frac{27}{30} * 100 = 90\%$$