

## **Práctica 1. Búsqueda en juegos**

### **Objetivos:**

- Comprender el funcionamiento de la búsqueda en juegos.
- Desarrollar una función de evaluación para un juego en concreto.
- Realizar un análisis entre los dos algoritmos implementados (minimax y alfabeta).

### ***Sesión 1: Introducción y entorno de trabajo. Diseño de la función de evaluación.***

En esta primera práctica de la asignatura se debe desarrollar un algoritmo de búsqueda en juegos para el juego Conecta-4. En este juego el jugador va colocando fichas que se deslizan por un tablero vertical para intentar conseguir 4 fichas en línea. Se empezará con el diseño de una función de evaluación adecuada para el juego. Con este objetivo se debe estudiar el modo de funcionamiento del juego y por qué interesa colocar la ficha en una determinada posición (por ejemplo, para evitar que el jugador contrario tenga cuatro fichas en línea). La función de evaluación debe proporcionar una medida numérica de lo buena que es una determinada configuración del tablero para la máquina.

### **Funcionamiento del juego**

Al ejecutar el juego aparecerá un tablero preparado para introducir las fichas. El juego está pensado para funcionar en modo máquina-persona. Empieza jugando la persona. Para colocar una ficha simplemente hay que pulsar con el ratón en la columna en la que se quiere deslizar la ficha. Si la columna tiene hueco se mostrará una ficha de color rojo en la posición correspondiente. A continuación, es el turno del ordenador. Mediante el algoritmo implementado se determina la posición adecuada y aparecerá una ficha amarilla en dicha posición. El juego continúa hasta que uno de los jugadores consigue colocar 4 fichas en línea o el tablero se llena.

El entorno proporcionado decide la jugada de la máquina simplemente barriendo el tablero de izquierda a derecha y buscando la primera columna disponible donde insertar la ficha.

```
//PSEUDOCODIGO//

function minimax(node, depth, isMaximizingPlayer, alpha, beta):

    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal
```

```
function minimax( node, depth, maximizingPlayer ) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := -∞
        for each child of node do
            value := max(value, minimax(child, depth - 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min( value, minimax( child, depth - 1, TRUE ) )
        return value
```

## Script

El entorno se ha desarrollado en Python usando la librería pygame. Pygame permite la creación de videojuegos en dos dimensiones, permite programar la parte multimedia (gráficos, sonido y manejo de eventos) de forma sencilla.

El entorno proporcionado está compuesto por 3 ficheros:

- **main:** es el fichero que hay que ejecutar para lanzar el entorno. Tiene el bucle principal de manejo del juego, así como el desarrollo del interfaz gráfico
- **tablero:** permite construir un objeto tablero que contiene las dimensiones del mismo, así como la matriz que representa las posiciones de las fichas.
- **algoritmo:** contiene el módulo `juega` con la llamada al algoritmo de búsqueda. Proporciona el módulo `busca` que recibe el tablero y un indicador de columna y devuelve la fila de la primera celda disponible de esa columna del tablero.

### Ejecución del proyecto

Ejecutar el fichero `main.py`

**Tarea a realizar en esta primera sesión:**

- Prueba y analiza el entorno. Averigua cómo acceder a una celda del tablero e investiga acerca de un diseño adecuado para la función de evaluación.
- Comienza el diseño del algoritmo minimax.

## Sesiones 2 y 3: Implementación del algoritmo minimax. Hito intermedio de entrega

El algoritmo minimax genera todos los nodos del árbol de búsqueda hasta la profundidad deseada, evalúa cada nodo hoja y asigna un valor a la raíz dependiendo de si es max o min. El algoritmo calcula el valor  $V(N)$  de un determinado nodo.


```
algoritmo minimax.  $V(N)$ 
Entrada: nodo N
Salida: valor de dicho nodo

Si N es nodo terminal entonces devolver  $f(N)$ 
sino
    generar todos los sucesores de N:  $N_1, N_2, \dots, N_b$ 
    Si N es MAX entonces devolver  $\max(V(N_1), V(N_2), \dots, V(N_b))$  fsi
    Si N es MIN entonces devolver  $\min(V(N_1), V(N_2), \dots, V(N_b))$  fsi
fsi
falgoritmo
```

En el código del `main` suministrado aparece la llamada al módulo `juega`. Este módulo se encuentra en el fichero `algoritmo` y es el que tiene que llamar al algoritmo de búsqueda.

El minimax debe actualizar el array `posición` (es uno de los parámetros de `juega`) con la fila y la columna en la que hay que colocar la ficha

```
posicion=[-1,-1]
juega(tablero, posicion)
tablero.setCelda(posicion[0], posicion[1], 2)
```



### Tareas a realizar en estas sesiones:

- Implementar el algoritmo minimax.
- Antes de la sesión 4 se debe entregar el proyecto con el minimax y la función de evaluación básica implementada hasta el momento

## Sesiones 4 y 5: Implementación el algoritmo alfa-beta

En estas sesiones se va a implementar el algoritmo de poda alfa-beta. Este algoritmo realiza podas de las ramas que no le llevan a obtener una mejor solución que la que ya tiene hasta el momento

```

Algoritmo  $\alpha - \beta . V(N, \alpha, \beta)$ 
Entrada: Nodo  $N$ , valores  $\alpha$  y  $\beta$ .
Salida: Valor minimax de dicho nodo.

Si  $N$  es nodo hoja entonces devolver  $f(N)$ .
sino
  Si  $N$  es nodo MAX entonces
    Para  $k = 1$  hasta  $b$  hacer
       $\alpha = \max[\alpha, V(N_k, \alpha, \beta)]$ 
      Si  $\alpha \geq \beta$  entonces devolver  $\beta$  FinSi.
      Si  $k = b$  entonces devolver  $\alpha$  FinSi.
    FinPara.
  sino
    Para  $k = 1$  hasta  $b$  hacer
       $\beta = \min[\beta, V(N_k, \alpha, \beta)]$ 
      Si  $\alpha \geq \beta$  entonces devolver  $\alpha$  FinSi.
      Si  $k = b$  entonces devolver  $\beta$  FinSi.
    FinPara.
  FinSi
FinSi

```

Al igual que con el algoritmo minimax, el alfa-beta debe actualizar el array posición con la fila y la columna en la que hay que colocar la ficha

#### Tareas a realizar en estas sesiones:

- Implementar el algoritmo alfa-beta
- Perfeccionamiento de la función de evaluación

## Sesión 6: Documentación y pruebas. Hito final

Esta última sesión está dedicada a realizar pruebas y terminar la documentación que se deberá haber ido elaborando de manera continua durante todo el desarrollo de la práctica.

La documentación es la parte más importante de la práctica (60%). **Como mínimo** debe contener:

- Explicación detallada de los algoritmos implementados
- Explicación de la función de evaluación diseñada
- Sección de experimentación en la que se describan las diferentes pruebas realizadas. 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. En esta sección se deben mostrar capturas de pantalla que muestren los experimentos realizados.
- Comparativa de tiempos y de número de nodos generados entre ambos algoritmos con varios niveles de profundidad del árbol de juego (p.e profundidad 2, 3 y 5).

### Entrega de la práctica

La fecha límite de entrega de la práctica es el 30 de octubre de 2022 a las 23:55h. La entrega se realizará a través de Moodle.

### Formato de entrega del proyecto para el hito 2 (entrega final)

La entrega debe consistir en un **fichero comprimido zip** con dos carpetas:

- /Fuente: todos los ficheros .py que constituyen la práctica
- /Doc: documentación en **pdf**

### !!!AVISO IMPORTANTE!!!

No cumplir cualquiera de las normas de formato/entrega puede suponer un suspenso en la práctica.

Recordad que **las prácticas son INDIVIDUALES** y NO se pueden hacer en parejas o grupos.

**Cualquier código copiado supondrá un suspenso de la práctica para todas las personas implicadas en la copia y, como indica el Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante (BOUA 9/12/2015) y el documento de Actuación ante copia en pruebas de evaluación de la EPS, se informará a la dirección de la Escuela Politécnica Superior para la toma de medidas oportunas.**

### Plan de entrega por hitos

Durante el periodo de ejecución de la práctica se realizarán dos hitos de entrega. Es obligatorio cumplir las fechas de las entregas correspondientes:

Hito	Entrega	Fecha tope
1 (intermedio)	Todos los ficheros .py	9 de octubre
2 (final)	Práctica completa siguiendo la estructura del formato de entrega	30 de octubre

- La no entrega del hito 1 en la fecha prevista supone una penalización del 20%.
- Para la entrega final se dejará el programa funcionando con el algoritmo alfa-beta.

**La nota de la práctica sufrirá una penalización de dos puntos si no se cumple rigurosamente con los requisitos de la entrega**