# PROYECTO DE PROGRAMACIÓN - PRIMER TRIMESTRE

## Descripción del proyecto

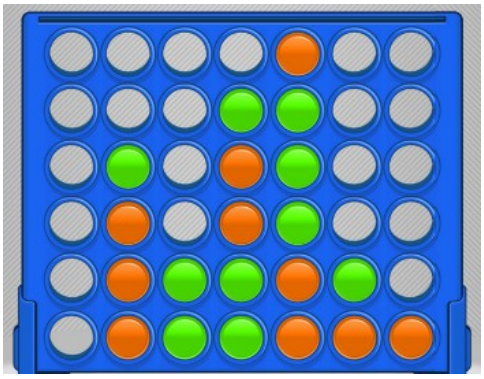
En este proyecto tendrás que poner en práctica los conocimientos adquiridos hasta el momento. El proyecto consiste en la realización de un juego de tablero. La descripción completa y detallada de la implementación del proyecto se encuentra en la siguiente sección.

Debes aplicar todas las buenas prácticas aprendidas:

- Validar todos los datos de entrada, tanto los introducidos por teclado por el usuario, como los parámetros de las funciones.
- Evitar código duplicado y reutilizar código mediante funciones.
- Nombrar las variables y funciones de acuerdo a las convenciones del lenguaje, y usar identificadores descriptivos y representativos de lo que almacena o realiza la variable/función.
- Seguir las especificaciones descritas en este guion sin cambiarlas, salvo justificación y previa consulta con el profesor.

## Juego del Conecta N (10 puntos)

El juego **Conecta 4** o **4 en línea** es un juego de tablero de 6 filas y 7 columnas en donde dos jugadores se turnan para colocar su ficha en una columna libre (https://es.wikipedia.org/wiki/Conecta_4). En la imagen se muestra un ejemplo del desarrollo del juego con fichas de colores. El tablero está colocado en vertical, de modo que tiene la peculiaridad de que al colocar las fichas en una columna, estas caen y se depositan justo encima de la última pieza de esa columna.



El **juego termina** en alguna de las circunstancias siguientes:

a) Cuando un jugador obtiene **4 fichas en línea** (horizontal, vertical o diagonal) gana la partida.
b) Cuando no quedan casillas libres, se produce un empate.

Vamos a realizar la implementación de este juego en **Python** con una configuración más flexible:

- El juego será **Conecta-N**, es decir, el número de fichas en línea necesarias para ganar la partida será definido por un parámetro. Si se desea, se puede establecer un valor mínimo y máximo en función del tamaño del tablero elegido por el usuario.
- El **tamaño del tablero será configurable**, por lo que se podrá indicar un número distinto de filas y columnas al tablero original, siempre y cuando este sea mayor o igual al original (6x7).
- Habrá **dos modos de juego: Jugador1 vs Jugador2** y **Jugador vs IA**.

La implementación en Python se realizará en la consola de comandos por lo que la representación del tablero y de las fichas se hará en modo texto con caracteres. Se debe realizar la implementación siguiendo las siguientes especificaciones.

### **(2 puntos)** Programa principal

La función ***main*** será la encargada de crear el **tablero utilizando el tipo List de Python** que representará una matriz de enteros inicializados a 0 (casilla vacía). Para representar las fichas de los jugadores (O) y (X) usaremos valores enteros. Para ello, definiremos las siguientes constantes para trabajar en el tablero a través de ellas.

- *CASILLA_VACIA=0;*
- *FICHA_CIRCULO=1;*
- *FICHA_EQUIS=2;*

Se deben incluir las siguientes consideraciones en el programa principal:

a) **(0.5 puntos)** El programa deberá solicitar el tamaño del tablero (mínimo 6 filas y 7 columnas) y el número de fichas en línea con el que se gana la partida (mínimo 4 y máximo el que permita el tablero). El programa funcionará de forma genérica independientemente de los valores seleccionados.

b) **(0.25 puntos)** El programa solicitará el modo de juego y el nombre de los jugadores para pedir por turnos que introduzcan la columna del tablero donde quieren colocar la ficha.

c) **(0.25 puntos)** El programa informará al usuario si la columna introducida es errónea (no existe o está ya ocupada) para que inserte una nueva columna.

d) **(0.5 puntos)** Si la columna introducida es correcta, se mostrará el tablero con la nueva ficha en su posición. La última ficha colocada debe aparecer resaltada en el tablero con un color distinto al resto de fichas.

e) **(0.5 puntos)** El programa continuará hasta que se produzca el final de la partida, bien porque algún jugador ha conseguido colocar N fichas en línea o más, bien porque no quedan casillas libres.

### **(6 puntos)** Las funciones del juego

En el fichero donde se implementa el programa principal se crearán las funciones necesarias para manejar la lógica del juego. En particular, se crearán al menos las siguientes funciones:

- **(0,5 puntos)** **mostrar_tablero(tablero)**. Visualizará el tablero en pantalla de forma apropiada para que el usuario lo entienda. Habrá que escribir O o X para representar las fichas de los jugadores, en lugar de valores numéricos. Las fichas de cada tipo aparecerán resaltadas con

algún color.

- **(0.5 puntos) mostrar_tablero_columna(tablero, columna_ultima_ficha)**. Igual que la función anterior, pero además recibe por parámetro la columna de la última ficha colocada para poder mostrarla de forma resaltada con un color distinto.
- **(0.5 puntos) colocar_ficha(tablero, ficha, columna)**. Coloca la *ficha* en la *columna* indicada (encima de las fichas existentes). Devuelve *True* si ha habido éxito y *False* si no.
- **(2 puntos) comprobar_linea(tablero, columna, numero_fichas_linea)**. Devuelve *True* si en el tablero hay *numero_fichas_linea* en línea (horizontal, vertical o diagonal), *False* en caso contrario. La comprobación se realizará desde la posición de la última ficha colocada, para ello se dispone del parámetro *columna*.
- **(2 puntos) fichas_en_linea(tablero, ficha, columna)**. Devuelve el número de fichas máximo en línea (horizontal, vertical o diagonal) que se harían si se colocara una ficha en esa columna. Esta función no coloca la ficha en esa columna. Esta función es útil para el modo de juego de la IA en el nivel 2 de dificultad.
- **(0,5 puntos) hay_casillas_libres(tablero)**. Devuelve *True* si en el tablero hay alguna casilla libre, o *False* si el tablero está completo.

**(2 puntos) El juego de la máquina.**

En el modo de juego Jugador vs IA se deberán programar **dos niveles de dificultad**. Estos niveles se elegirán en el programa principal y tendrán las siguientes características.

- **(0,5 puntos)** Nivel 1: El sistema coloca la ficha de manera aleatoria entre las columnas disponibles.
- **(1,5 puntos)** Nivel 2: El sistema debe actuar aplicando las siguientes reglas en orden de prioridad:
  1. Colocar la ficha en la columna donde puede ganar la partida (en el tablero hay *n-1* fichas en línea de la máquina).
  2. Colocar la ficha en la columna donde evita que el Jugador gane la partida (en el tablero hay *n-1* fichas en línea del jugador).
  3. Colocar la ficha en la columna donde la máquina puede obtener más fichas en línea. En caso de que haya varias opciones, se podrá decidir de manera aleatoria.
  4. Colocar la ficha en una columna aleatoria si no se cumplen las reglas anteriores.

  Se otorgará una puntuación de 0,5 puntos por cada regla aplicada en el nivel 2, salvo la regla 4. Por ejemplo, si se aplica la regla 1 y posteriormente la regla 4 se otorgarán 0,5 puntos.

Si es necesario, las funciones anteriores se pueden dividir en funciones más pequeñas (divide y vencerás) que permitan reutilizar al máximo el código creado. Esta reutilización del código es valorada en la corrección de la tarea.

## Entrega de la tarea

Se comprimirá en un único fichero en formato .ZIP la carpeta juegoconectaN que contendrá todo el código necesario para ejecutar el juego. El fichero se subirá al buzón de la tarea en la plataforma Moodle.

El archivo se nombrará siguiendo las siguientes pautas:
**Apellido1_Apellido2_Nombre_PYTHON_ProyectoPrimerTrimestre.zip**

## Rúbrica de evaluación

Cada apartado puntuable del proyecto se valorará con la siguiente rúbrica.

| # | Criterio | Porcentaje |
|---|---|---|
| **1** | El programa/función implementado no cumple los requisitos, no soluciona de forma algorítmica el ejercicio o las soluciones obtenidas por el programa no son las esperadas, el código no compila o contiene errores. | 0% |
| **2** | El programa/función implementado soluciona de forma algorítmica el ejercicio pero falla con datos de entrada no permitidos. | 25% |
| **3** | El programa/función implementado tiene fallos inesperados en situaciones específicas o concretas, es decir, falla para un determinado caso o valor de entrada, pero en general el resultado obtenido es válido. | 50% |
| **4** | El programa/función implementado cumple los requerimientos pero:<br>• El código no es legible (identificadores inadecuados, sin comentarios relevantes).<br>• El código no está bien estructurado (código duplicado, ineficiente, innecesario). | 75% |
| **5** | El programa/función implementado se ajusta perfectamente a la especificación:<br>• Se validan los datos de entrada y el resultado obtenido es válido para cualquier dato de entrada.<br>• El código es modular y se emplean funciones/métodos adecuadamente.<br>• El código es legible y usa comentarios relevantes y/o Javadoc. | 100% |

## Resultados de aprendizaje y criterios de evaluación relacionados

En esta actividad se evalúan los siguientes resultados de aprendizaje con los criterios de evaluación que se relacionan para cada uno de ellos:

- RA 1. Identifica las estructuras de control en Python relacionándolas con aplicaciones reales.
    a) Se han identificado las estructuras de control que permiten modificar el flujo de las instrucciones.
    b) Se han representado en un diagrama de flujo gráfico las estructuras de control.
    c) Se han analizado la importancia de las condiciones en cada estructura de control.
    d) Se han tenido en cuenta la importancia de los sangrados en las estructuras de control.
    e) Se han escrito bloques de control secuencial.
    f) Se han escrito bloques de control de selección.
    g) Se han escrito bloques de control de repetición.
- RA 2. Reconoce las sentencias condicionales en Python aplicándolas a la resolución de problemas que impliquen toma de decisiones.
    a) Se ha interpretado el concepto de sentencia condicional.
    b) Se han identificado las partes de las que consta una sentencia condicional.
    c) Se ha aplicado correctamente el sangrado.
    d) Se ha aplicado la ejecución condicional y control de variables.
    e) Se han interpretado el funcionamiento de las sentencias condicionales.
    f) Se han aplicado correctamente las sentencias condicionales.
    g) Se han interpretado y aplicado correctamente las anidaciones.
    h) Se aplica correctamente la sintaxis a aplicar en estructuras compactas.
    i) Se han escrito bloques de programas utilizando sentencias condicionales.

j) Se han escrito bloques de programas utilizando sentencias condicionales anidadas.

- RA 3. Utiliza sentencias iterativas analizando las necesidades del código para resolver un problema.
  a) Se ha interpretado el concepto de sentencia iterativa.
  b) Se ha diferenciado entre estructuras condicionales e iterativas.
  c) Se ha verificado el funcionamiento de las sentencias iterativas.
  d) Se han aplicado las sentencias iterativas de acuerdo a las necesidades.
  e) Se han escrito bloques de programas utilizando los bucles «for» y «while».
  f) Se han interpretado y aplicado los anidamientos de estructuras.
- RA 4. Aplica funciones de Python de distintos tipos mejorando la eficiencia del programa.
  a) Se comprende la necesidad de usar funciones de Python y sus ventajas.
  b) Se ha escrito código que incluya funciones Build-in de Python.
  c) Se ha escrito un programa con funciones definidas por la propia persona usuaria.
  d) Se aplican correctamente las funciones lambda en un programa de Python.
  e) Se han creado funciones recursivas partiendo de funciones definidas anteriormente por la persona usuaria.
- RA 5. Crea arquitectura de código de forma eficiente y escribe código robusto.
  a) Se ha diferenciado entre el concepto de excepción y los errores de sintaxis.
  b) Se ha escrito instrucciones de captura de excepciones.
  c) Se han capturado y tratado excepciones.
  d) Se han tratado excepciones.
  e) Se han realizado depuraciones de excepciones correctamente.
  f) Se han escrito bloques de código robusto utilizando las sentencias adecuadas.