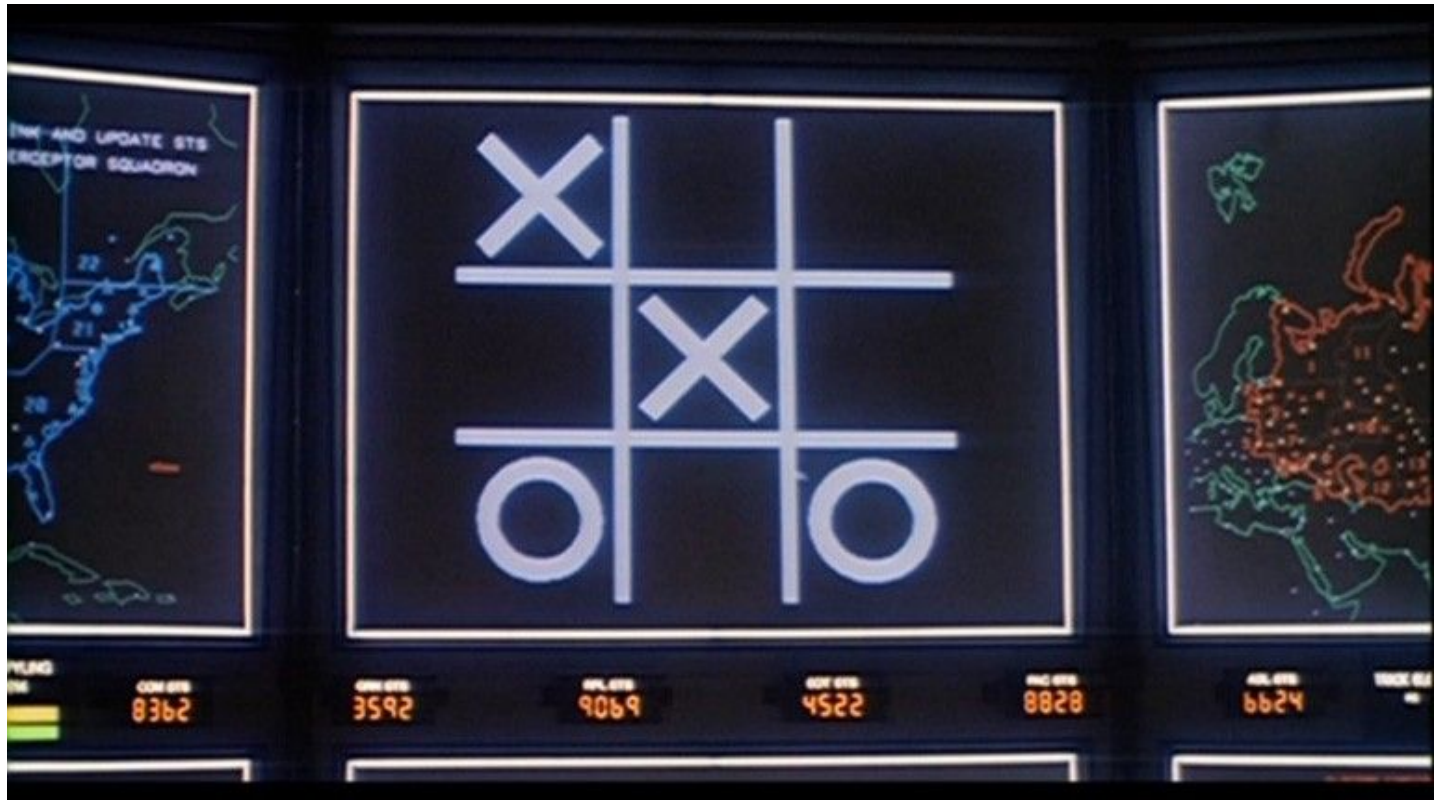
Práctica 2

OXO - inteligencia débil y fuerte

Profesor:
David Campoy Miñarro

Reglas del videojuego OXO

El juego OXO, también conocido como Tic-Tac-Toe, es un juego de mesa clásico para dos jugadores que se juega en un tablero de 3x3. El objetivo del juego es completar una fila, columna o diagonal con tres fichas del mismo jugador.



La inteligencia artificial *Joshua* de la película **Juegos de Guerra** (1983) aprende las reglas de la guerra jugando a miles de partidas a OXO.

OXO con inteligencia débil

```
import random

# Inicializar el tablero
board = [""] * 9
player_turn = True
game_over = False

# Función para imprimir el tablero
def print_board(board):
    for i in range(0, 9, 3):
        print(board[i], "|", board[i+1], "|", board[i+2])
        if i < 6:
            print("---------")

# Función para verificar si un jugador ha ganado
def check_winner(board, player):
    for line in [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]:
        if all(board[i] == player for i in line):
            return True
    return False

# Función para que la IA haga un movimiento aleatorio
def ai_move(board):
    empty_cells = [i for i in range(9) if board[i] == ""]
    if empty_cells:
        return random.choice(empty_cells)
    else:
        return None
```


Profesor: David Campoy Miñarro

```
# Ciclo principal del juego
while not game_over:
    print_board(board)
    if player_turn:
        print("Tu turno (X)")
        move = input("Elige una casilla (0-8): ")
        if move.isdigit() and 0 <= int(move) < 9 and board[int(move)] == "":
            board[int(move)] = "X"
            player_turn = False
    else:
        print("Turno de la IA (O)")
        ai_position = ai_move(board)
        if ai_position is not None:
            board[ai_position] = "O"
            player_turn = True
    if check_winner(board, "X"):
        print_board(board)
        print("¡Ganaste!")
        game_over = True
    elif check_winner(board, "O"):
        print_board(board)
        print("¡La IA ganó!")
        game_over = True
    elif "" not in board:
        print_board(board)
        print("Empate")
        game_over = True
print("Fin del juego")
```

OXO con inteligencia ¿fuerte?

Este nuevo algoritmo va a simular 10.000 partidas y va a aprender de los errores.
El programa va a utilizar el método de aprendizaje *Q-Learning*.

El Q-learning es un algoritmo de aprendizaje por refuerzo que se utiliza para tomar decisiones secuenciales en un entorno para maximizar una recompensa acumulada a lo largo del tiempo. Es uno de los algoritmos más utilizados para abordar problemas de toma de decisiones en inteligencia artificial.

```
Turno de la IA (O)
X X O
  O

Tu turno (X)
Elige una casilla (0-8): 3
X X O
X O

Turno de la IA (O)
X X O
X O
O
¡La IA ganó!
```

Ecuación de Bellman:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V^{\pi}(s') \right]$$

OXO con inteligencia fuerte

```
#!pip install numpy
import random
# Inicializar el tablero
board = [""] * 9
# Parámetros de Q-learning
learning_rate = 0.1
discount_factor = 0.9
# Crear una tabla Q para el agente (en este caso, el jugador "X")
q_table = {}
# Función para codificar el estado del tablero a una cadena
def encode_state(board):
    return "".join(board)
# Función para que la IA haga un movimiento basado en Q-learning
def ai_move(board):
    state = encode_state(board)
    if state not in q_table:
        q_table[state] = [0.0] * 9  # Inicializar valores Q para el estado
    legal_moves = [i for i, cell in enumerate(board) if cell == ""]
    if random.uniform(0, 1) < exploration_prob:
        return random.choice(legal_moves)
    else:
        return max(legal_moves, key=lambda move: q_table[state][move])
```


Profesor: David Campoy Miñarro

```
# Función para actualizar la tabla Q después de una jugada
def update_q_table(state, action, reward, next_state):
    if state not in q_table:
        q_table[state] = [0.0] * 9
    if next_state not in q_table:
        q_table[next_state] = [0.0] * 9
    q_table[state][action] = q_table[state][action] + learning_rate *
(reward + discount_factor * max(q_table[next_state]) -
q_table[state][action])

# Función para verificar si un jugador ha ganado
def check_winner(board, player):
    for line in [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7],
[2, 5, 8], [0, 4, 8], [2, 4, 6]]:
        if all(board[i] == player for i in line):
            return True
    return False

# Parámetro de exploración
exploration_prob = 0.1
```

```
# Ciclo principal del juego
for _ in range(10000):
    board = [""] * 9
    state = encode_state(board)
    while True:
        if "X" not in board and "O" not in board:
            action = random.randint(0, 8)
        else:
            action = ai_move(board)
        if board[action] == "":
            board[action] = "X"
        next_state = encode_state(board)
        if check_winner(board, "X"):
            reward = 1
        elif check_winner(board, "O"):
            reward = -1
        elif "" not in board:
            reward = 0
        else:
            reward = 0  # Juego en curso
        update_q_table(state, action, reward, next_state)
        if check_winner(board, "X"):
            break
        state = next_state
```

```
# Ahora, puedes utilizar la tabla Q entrenada para jugar contra la IA
board = [""] * 9
player_turn = True
while True:
    if player_turn:
        print("Tu turno (X)")
        move = int(input("Elige una casilla (0-8): "))
        if board[move] == "":
            board[move] = "X"
            player_turn = False
    else:
        print("Turno de la IA (O)")
        action = ai_move(board)
        board[action] = "O"
        player_turn = True
    # Mostrar el tablero
    for i in range(0, 9, 3):
        print(" ".join(board[i:i+3]))

    if check_winner(board, "X"):
        print("¡Ganaste!")
        break
    elif check_winner(board, "O"):
        print("¡La IA ganó!")
        break
    elif "" not in board:
        print("Empate")
        break
```

¿Has notado la diferencia?
¿Es más inteligente?

Modifica el hiperparámetro:

```
for _ in range(10000):
```

y comprueba cómo varía su inteligencia.

Investiga sobre el método Q-Learning.

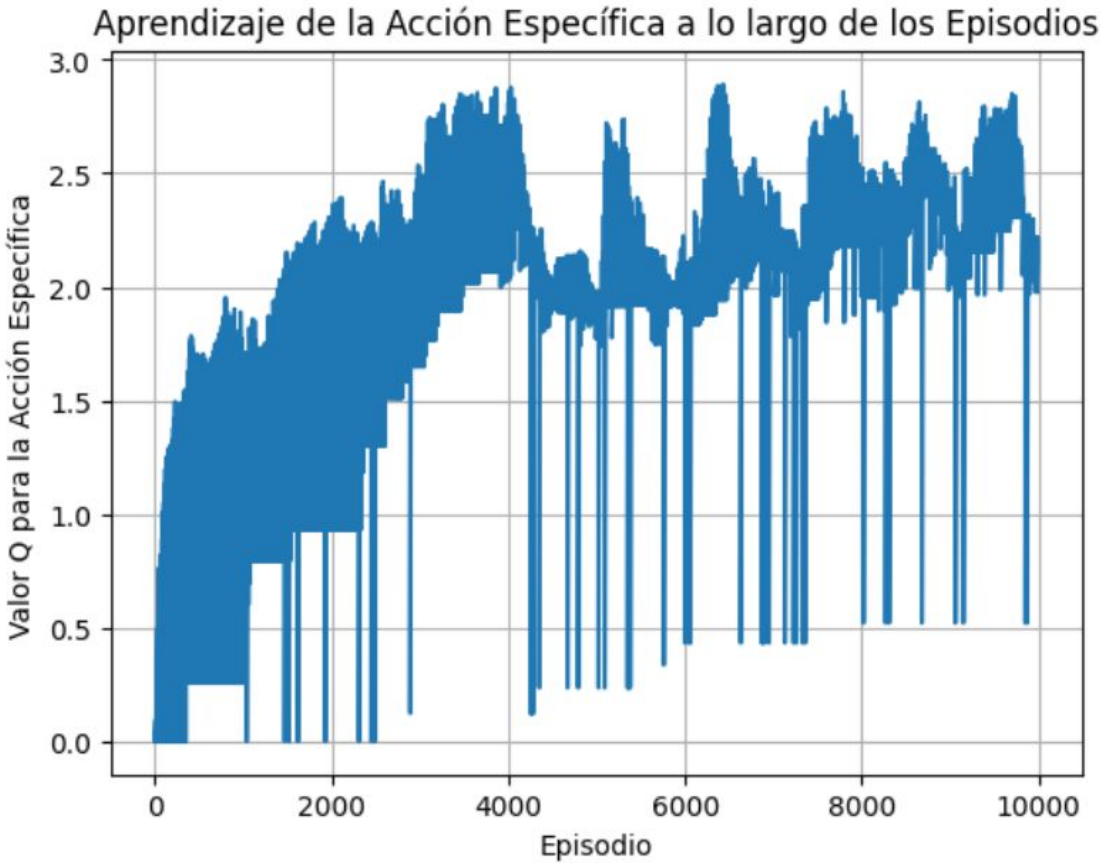
¿Pára que sirve los hiperparámetros?

```
learning_rate = 0.1
discount_factor = 0.9
```


Profesor: David Campoy Miñarro

Evolución de aprendizaje

```
learning_rate = 0.1
discount_factor = 0.9
```




```
        if check_winner(board, "X"):
            break

        state = next_state

        # Almacena los valores Q para una acción específica en función de los episodios
        if episode % 10 == 0:
            action_to_track = 0  # Cambiar a la acción que deseas rastrear
            episodes.append(episode)
            q_values.append(q_table[state][action_to_track])

# Graficar los valores Q en función de los episodios
plt.plot(episodes, q_values)
plt.title('Aprendizaje de la Acción Específica a lo largo de los Episodios')
plt.xlabel('Episodio')
plt.ylabel('Valor Q para la Acción Específica')
plt.grid()
plt.show()
```

¿Y con una red neuronal será más inteligente?

```python
import random
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Inicializar el tablero
board = [""] * 9

# Función para codificar el estado del tablero a una cadena
def encode_state(board):
    return "".join(board)

# Función para crear una red neuronal simple
def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(9,)),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(9, activation='linear')  # 9 salidas para cada casilla
    ])
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

```python
# Función para que la IA haga un movimiento basado en la red neuronal
def ai_move(board, model):
    state = np.array([1 if cell == "X" else -1 if cell == "O" else 0 for cell in board])
    q_values = model.predict(np.array([state]))[0]
    legal_moves = [i for i, cell in enumerate(board) if cell == ""]
    q_values_available = [q_values[i] for i in legal_moves]
    return legal_moves[np.argmax(q_values_available)]

# Función para verificar si un jugador ha ganado
def check_winner(board, player):
    for line in [[0, 1, 2], [3, 4, 5], [6, 7, 8], [0, 3, 6], [1, 4, 7], [2, 5, 8], [0, 4, 8], [2, 4, 6]]:
        if all(board[i] == player for i in line):
            return True
    return False
```

```python
# Ciclo principal del juego
model = create_model()
errors = []  # Lista para almacenar los errores en cada iteración
for _ in range(100):
    board = [""] * 9
    state = np.array([1 if cell == "X" else -1 if cell == "O" else 0 for cell in board])
    while True:
        if "X" not in board and "O" not in board:
            action = random.randint(0, 8)
        else:
            action = ai_move(board, model)
        if board[action] == "":
            board[action] = "X"
        next_state = np.array([1 if cell == "X" else -1 if cell == "O" else 0 for cell in board])
        if check_winner(board, "X"):
            reward = 1
        elif check_winner(board, "O"):
            reward = -1
        elif "" not in board:
            reward = 0
        else:
            reward = 0  # Juego en curso
        target = model.predict(np.array([state]))[0]
        target[action] = reward
        error = model.fit(np.array([state]), np.array([target]), epochs=1, verbose=0).history['loss'][0]
        errors.append(error)
        if check_winner(board, "X"):
            break
        state = next_state
```

```
# Generar la gráfica de evolución del error
plt.plot(errors)
plt.xlabel('Iteración')
plt.ylabel('Error')
plt.title('Evolución del error en el aprendizaje')
plt.show()
```

```python
# Ahora, puedes utilizar la red neuronal entrenada para jugar contra la IA
board = [""] * 9
player_turn = True
while True:
    if player_turn:
        print("Tu turno (X)")
        move = int(input("Elige una casilla (0-8): "))
        if board[move] == "":
            board[move] = "X"
            player_turn = False
    else:
        print("Turno de la IA (O)")
        action = ai_move(board, model)
        board[action] = "O"
        player_turn = True
    # Mostrar el tablero
    for i in range(0, 9, 3):
        print(" ".join(board[i:i+3]))

    if check_winner(board, "X"):
        print("¡Ganaste!")
        break
    elif check_winner(board, "O"):
        print("¡La IA ganó!")
        break
    elif "" not in board:
        print("Empate")
        break
```