ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS
INGENIERÍA EN SOFTWARE

Profesor: Ing. Myriam Hernández
Materia: Inteligencia Artificial y Aprendizaje Automático
Período académico: 2021-B
Grupo: GR1

Tema: Proyecto TIC-TAC-TOE

La función initial_state regresa el estado inicial. Para este problema se escoge representar el tablero como una lista de tres listas que representan las 3 filas del tablero, donde cada lista interna contiene tres valores que son x, o o EMPTY.

Las siguientes funciones deben ser implementadas:

La función player debe tomar un estado del tablero como entrada y devolver el turno de qué jugador es. (ya sea X u O).

- En el estado inicial del juego, X obtiene el primer movimiento. Posteriormente, el jugador alterna con cada movimiento adicional.
- Cualquier valor devuelto es aceptable si se proporciona una placa de terminales como entrada (es decir, el juego ya ha terminado).

```
"""
Tic Tac Toe Player
"""
import random
from copy import deepcopy

X = "X"
O = "O"
EMPTY = None
currentPlayer=""

def initial_state():···

def player(board):
    """
    Returns player who has the next turn on a board.
    """
    global currentPlayer
    XC = 0
    OC = 0

    for row in board:
      XC += row.count(X)
      OC += row.count(O)
    if XC > OC:
      currentPlayer=O
    else:
      currentPlayer=X
    return currentPlayer
```

Se verifica el número de X y de O en el tablero para obtener el siguiente turno.

La función actions debe devolver un conjunto de todas las acciones posibles que se pueden tomar en un tablero dado.

- Cada acción debe representarse como una tupla (i, j) donde i corresponde a la fila del movimiento (0, 1 o 2) y j corresponde a qué celda de la fila corresponde al movimiento (también 0, 1, o 2).
- Los movimientos posibles son las casillas del tablero que aún no tienen una X o una O en ellas.
- Cualquier valor devuelto es aceptable si se proporciona una placa de terminales como entrada.

```
def actions(board):
    """
    Returns set of all possible actions (i, j) available on the board.
    """
    moves = set()
    for i in range(3):
      for j in range(3):
        if board[i][j] == EMPTY:
          moves.add((i, j))

    return moves
```

La función result toma una placa y una acción como entrada, y debería devolver un nuevo estado de placa, sin modificar la placa original.

- Si la acción no es una acción válida para la junta, su programa debe generar una excepción.
- El estado del tablero devuelto debe ser el tablero que resultaría de tomar el tablero de entrada original y dejar que el jugador al que le toca hacer su movimiento en la celda indicada por la acción de entrada.
- Importante destacar que la placa original debe dejarse sin modificar: ya que Minimax finalmente requerirá considerar muchos estados diferentes de la placa durante su cálculo. Esto significa que simplemente actualizar una celda en la placa en sí no es una implementación correcta de la función de resultado. Es probable que primero desee hacer una copia detallada del tablero antes de realizar cualquier cambio.

```
def result(board, action):
    """
    Returns the board that results from making move (i, j) on the board.
    """
    nBoard = deepcopy(board)
    nBoard[action[0]][action[1]]=player(board)
    return nBoard
```

La función winner debe aceptar un tablero como entrada y devolver el ganador del tablero si hay uno.

- Si el jugador X ha ganado el juego, su función debe devolver X. Si el jugador O ha ganado el juego, su función debería devolver O.
- Puede ganar el juego con tres de sus movimientos seguidos horizontal, vertical o diagonalmente.
- Suponer que habrá como máximo un ganador (es decir, ningún tablero nunca tener a ambos jugadores con tres en fila, ya que eso sería un estado de tablero inválido).
- Si no hay un ganador del juego (ya sea porque el juego está en progreso o porque terminó en un empate), la función debería devolver Ninguno.

```python
def winner(board):
    """
    Returns the winner of the game, if there is one.
    """
    rowWinner = checkRows(board)
    columnWinner = checkColumns(board)
    diagonalWinner = checkDiagonals(board)
    # Get the winner
    if rowWinner:
        return  rowWinner
    elif columnWinner:
        return  columnWinner
    elif diagonalWinner:
        return diagonalWinner
    else:
        return  None
```

```python
def checkRows(board):
    for row in board:
      if row.count(X) == 3:
        return X
      if row.count(0) == 3:
        return 0
```

```python
def checkColumns(board):
    for j in range(3):
        k=0
        l=0
        for i in range(3):
            if(board[i][j]==X):
                k+=1
            elif(board[i][j]==0):
                l+=1

        if k==3:
            return X
        if l==3:
            return 0
```

```
def checkDiagonals(board):
    j = 2
    k=0
    l=0
    for i in range(3):

        if(board[i][i]==X):
            k+=1
        elif(board[i][i]==0):
            l+=1
    if k==3:
        return X
    if l==3:
        return 0
    k=0
    l=0

    for i in range(3):
        if(board[i][j]==X):
            k+=1
        elif(board[i][j]==0):
            l+=1
        j -= 1
    if k==3:
        return X
    if l==3:
        return 0
```

La función terminal debe aceptar una placa como entrada y devolver un valor booleano indicando si el juego ha terminado.

- Si el juego termina, ya sea porque alguien ha ganado el juego o porque todas las celdas se han llenado sin que nadie gane, la función debería devolver True.
- De lo contrario, la función debería devolver False si el juego aún está en progreso.

```
def terminal(board):
    """
    Returns True if game is over, False otherwise.
    """
    if winner(board) or not actions(board):
      return True
    else:
      return False
```

La función utility debe aceptar una placa de terminales como entrada y dar salida a la utilidad de la placa.

- Si X ganó el juego, la utilidad es 1. Si O ganó el juego, la utilidad es -1. Si el juego ha terminado en empate, la utilidad es 0.
- Puede asumir que la utilidad solo se llamará en una placa si el terminal (placa) es Verdadero.

```
def utility(board):
    """
    Returns 1 if X has won the game, -1 if O has won, 0 otherwise.
    """
    win = winner(board)
    if win == X:
        return 1
    elif win == O:
        return -1
    else:
        return 0
```

La función minimax debe tomar un tablero como entrada y devolver el movimiento óptimo para que el jugador se mueva en ese tablero.

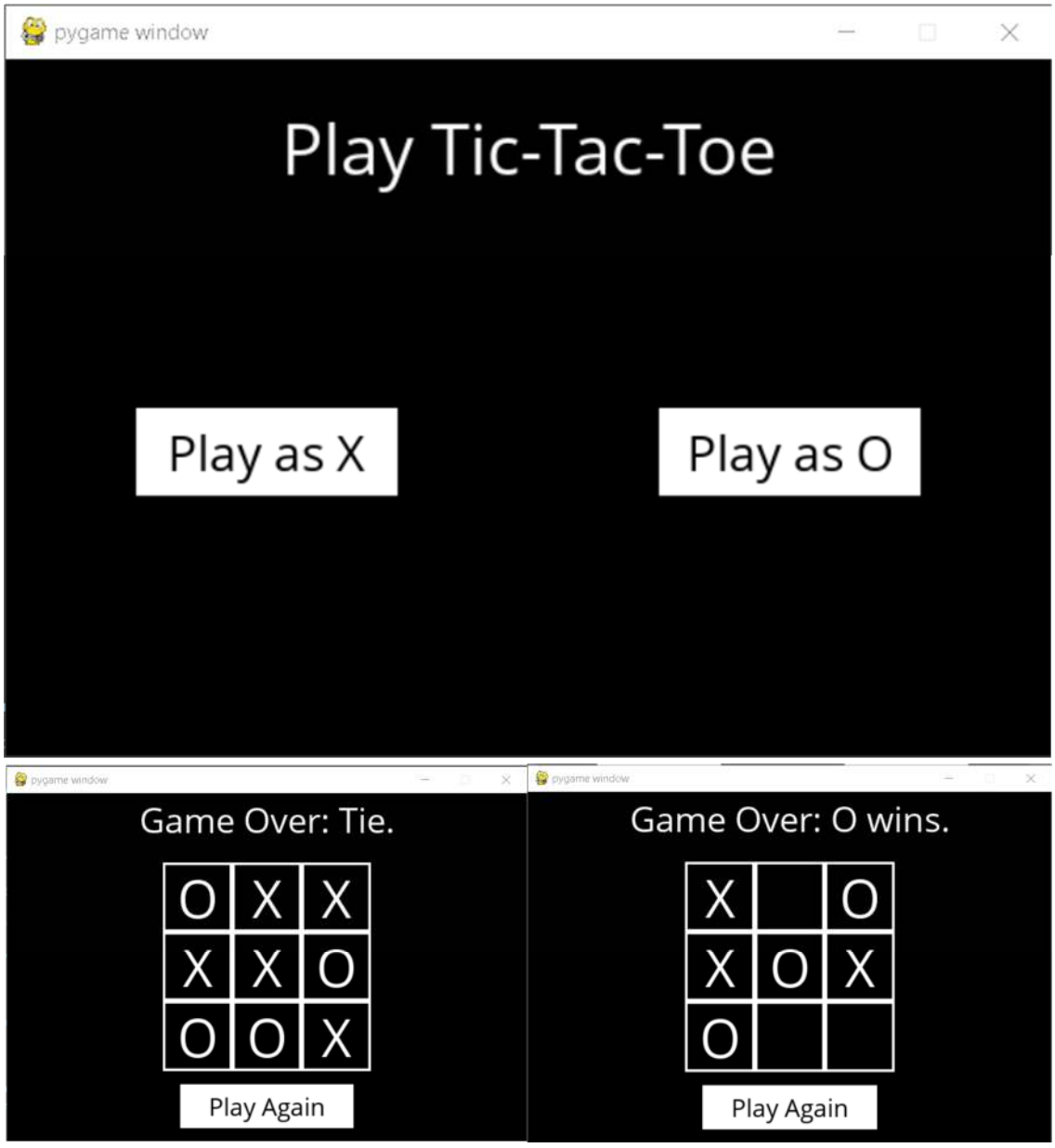
- El movimiento devuelto debe ser la acción óptima (i, j) que es una de las acciones permitidas en el tablero. Si varios movimientos son igualmente óptimos, cualquiera de esos movimientos es aceptable.
- Si la placa es una placa de terminales, la función minimax debería devolver None.

```
def minimax(board):
    """
    Returns the optimal action for the current player on the board.
    """
    if terminal(board):
      return None
    if player(board) == 'X':
      bestMove = maximize(board,1)[1]
      return bestMove
    else:
      bestMove = minimize(board,-1)[1]
      return bestMove
```

```python
def maximize(board, bestMin):
    if terminal(board):
        return (utility(board), None)

    value = -1
    bestAction = None
    actionSet = actions(board)

    while len(actionSet) > 0:
        action = random.choice(tuple(actionSet))
        actionSet.remove(action)
        if bestMin <= value:
            break
        minPlayerResult = minimize(result(board, action), value)
        if minPlayerResult[0] > value:
            bestAction = action
            value = minPlayerResult[0]

    return (value, bestAction)
```

```python
def minimize(board, bestMax):
    if terminal(board):
        return (utility(board), None)

    value = 1
    bestAction = None
    actionSet = actions(board)

    while len(actionSet) > 0:
        action = random.choice(tuple(actionSet))
        actionSet.remove(action)
        if bestMax >= value:
            break
    maxPlayerResult = maximize(result(board, action), value)

    if maxPlayerResult[0] < value:
        bestAction = action
        value = maxPlayerResult[0]

    return (value, bestAction)
```

Ejecución



Conclusiones y recomendaciones:

Se puede mencionar que, si bien el algoritmo minimax implementado funciona correctamente, este posee ciertas fallas con un tipo de posición de fichas específico, puesto que ocupa un espacio que deja la posibilidad de ganar al jugador, sin embargo, en el resto de casos su funcionamiento es el adecuado para los objetivos del proyecto.