Importing the library

```
import numpy as np
import matplotlib.pyplot as plt
import pickle


## Initialization
BOARD_ROWS=3
BOARD_COLS=3
BOARD_SIZE=BOARD_ROWS*BOARD_COLS


class State:
  def __init__(self,p1,p2):
    self.board=np.zeros((BOARD_ROWS,BOARD_COLS))
    self.p1=p1
    self.p2=p2
    self.isEnd=False
    self.boardHash=None
    self.playerSymbol=1
    #Board State
  def getHash(self):
    self.boardHash=str(self.board.reshape(BOARD_SIZE))
    return self.boardHash
  def availablePositions(self):
    positions=[]
    for i in range(BOARD_ROWS):
      for j in range(BOARD_COLS):
        if self.board[i,j]==0:
          positions.append((i,j))
    return positions
  def updateState(self,position):
    self.board[position]=self.playerSymbol
    if self.playerSymbol==1:
      self.playerSymbol=-1
    else:
      self.playerSymbol=1
  ##Check Winner
  def winner(self):
    for i in range(BOARD_ROWS):
      if sum(self.board[i,:])==3:
        self.isEnd=True
        return 1
      if sum(self.board[i,:])==-3:
        self.isEnd=True
        return -1
    for i in range(BOARD_COLS):
      if sum(self.board[:,i])==3:
        self.isEnd=True
        return 1
```

```
      if sum(self.board[i,:])==-3:
        self.isEnd=True
        return -1
    #Diagonal
    diag_sum1=sum([self.board[i,i] for i in range(BOARD_COLS)])
    diag_sum2=sum([self.board[i,BOARD_COLS-i-1] for i in range(BOARD_COLS)])
    diag_sum=max(abs(diag_sum1),abs(diag_sum2))
    if diag_sum==3:
      self.isEnd=True
      if diag_sum1==3 or diag_sum2==3:
        return 1
      else:
        return -1
    if len(self.availablePositions())==0:
      self.isEnd=True
      return 0
    self.isEnd=False
    return None
  def giveReward(self):
    result=self.winner()
    if result==1:
      self.p1.feedReward(1)
      self.p2.feedReward(0)
    elif result==-1:
      self.p1.feedReward(0)
      self.p2.feedReward(1)
    else:
      self.p1.feedReward(0.1)
      self.p2.feedReward(0.5)
  # board reset
  def reset(self):
    self.board = np.zeros((BOARD_ROWS, BOARD_COLS))
    self.boardHash = None
    self.isEnd = False
    self.playerSymbol = 1
  def play(self,rounds=100):
    for i in range(rounds):
      if i%1000==0:
        print("Round {}".format(i))
      while not self.isEnd:
        positions=self.availablePositions()
        p1_action=self.p1.chooseAction(positions,self.board,self.playerSymbol)
        self.updateState(p1_action)
        board_hash=self.getHash()
        self.p1.addState(board_hash)
        win=self.winner()
        if win is not None:
          #self.showBoard()
          self.giveReward()
          self.p1.reset()
          self.p2.reset()
          self.reset()
          break
        else:
          positions=self.availablePositions()
```

```
            p2_action = self.p2.chooseAction(positions, self.board, self.playerSymbol)
            self.updateState(p2_action)
            board_hash = self.getHash()
            self.p2.addState(board_hash)
            win = self.winner()
            if win is not None:
              self.giveReward()
              self.p1.reset()
              self.p2.reset()
              self.reset()
              break
        # play with human
    def play2(self):
      while not self.isEnd:
        positions = self.availablePositions()
        p1_action = self.p1.chooseAction(positions, self.board, self.playerSymbol)
        self.updateState(p1_action)
        self.showBoard()
        win = self.winner()
        if win is not None:
          if win == 1:
            print(self.p1.name, "wins!")
          else:
            print("tie!")
          self.reset()
          break

        else:
          positions = self.availablePositions()
          p2_action = self.p2.chooseAction(positions)

          self.updateState(p2_action)
          self.showBoard()
          win = self.winner()
          if win is not None:
            if win == -1:
              print(self.p2.name, "wins!")
            else:
              print("tie!")
              self.reset()
              break

    def showBoard(self):
      for i in range(0, BOARD_ROWS):
        print('-------------')
        out = '| '
        for j in range(0, BOARD_COLS):
          if self.board[i, j] == 1:
            token = 'x'
          if self.board[i, j] == -1:
            token = 'o'
          if self.board[i, j] == 0:
            token = ' '
          out += token + ' | '
        print(out)
```

```
        print('-------------')



    ##Class player
    class Player:
        def __init__(self, name, exp_rate=0.2):
            self.name = name
            self.states = []
            self.lr = 0.1
            self.exp_rate = exp_rate
            self.decay_gamma = 0.9
            self.states_value = {}

        def getHash(self, board):
            boardHash = str(board.reshape(BOARD_COLS * BOARD_ROWS))
            return boardHash

        def chooseAction(self, positions, current_board, symbol):
            if np.random.uniform(0, 1) <= self.exp_rate:
                idx = np.random.choice(len(positions))
                action = positions[idx]
            else:
                value_max = -999
                for p in positions:
                    next_board = current_board.copy()
                    next_board[p] = symbol
                    next_boardHash = self.getHash(next_board)
                    value = 0 if self.states_value.get(next_boardHash) is None else self.state
                    #print("value", value)
                    if value >= value_max:
                        value_max = value
                        action = p
            return action
        def addState(self, state):
            self.states.append(state)

        def feedReward(self, reward):
            for st in reversed(self.states):
                if self.states_value.get(st) is None:
                    self.states_value[st] = 0
                self.states_value[st] += self.lr * (self.decay_gamma * reward - self.states_va
                reward = self.states_value[st]

        def reset(self):
            self.states = []

        def savePolicy(self):
            fw = open('policy_' + str(self.name), 'wb')
            pickle.dump(self.states_value, fw)
            fw.close()

        def loadPolicy(self, file):
            fr = open(file, 'rb')
```

```
            self.states_value = pickle.load(fr)
            fr.close()


    class HumanPlayer:
        def __init__(self, name):
            self.name = name

        def chooseAction(self, positions):
            while True:
                row = int(input("Input your action row:"))
                col = int(input("Input your action col:"))
                action = (row, col)
                if action in positions:
                    return action
        def addState(self, state):
            pass
        def feedReward(self, reward):
            pass

        def reset(self):
            pass




    if __name__ == "__main__":
        # training
        p1 = Player("p1")
        p2 = Player("p2")

        st = State(p1, p2)
        print("training...")
        st.play(50000)

        # play with human
        p1 = Player("computer", exp_rate=0)
        p1.loadPolicy("/content/drive/MyDrive/policy_p1")

        p2 = HumanPlayer("human")

        st = State(p1, p2)
        st.play2()
```

```
 training...
 Round 0
 Round 1000
 Round 2000
 Round 3000
 Round 4000
 Round 5000
 Round 6000
 Round 7000
 Round 8000
 Round 9000
```

```
Round 10000
Round 11000
Round 12000
Round 13000
Round 14000
Round 15000
Round 16000
Round 17000
Round 18000
Round 19000
Round 20000
Round 21000
Round 22000
Round 23000
Round 24000
Round 25000
Round 26000
Round 27000
Round 28000
Round 29000
Round 30000
Round 31000
Round 32000
Round 33000
Round 34000
Round 35000
Round 36000
Round 37000
Round 38000
Round 39000
Round 40000
Round 41000
Round 42000
Round 43000
Round 44000
Round 45000
Round 46000
Round 47000
Round 48000
Round 49000
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   |   |   |
```

```
# play with human
p1 = Player("computer", exp_rate=0)
p1.loadPolicy("/content/drive/MyDrive/policy_p1")

p2 = HumanPlayer("human")

st = State(p1, p2)
st.play2()
```

```
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   |   |   |
-------------
Input your action row:0
Input your action col:1
-------------
|   | o |   |
-------------
|   | x |   |
-------------
|   |   |   |
-------------
-------------
|   | o | x |
-------------
|   | x |   |
-------------
|   |   |   |
-------------
Input your action row:2
Input your action col:0
-------------
|   | o | x |
-------------
|   | x |   |
-------------
| o |   |   |
-------------
-------------
|   | o | x |
-------------
|   | x |   |
-------------
| o |   | x |
-------------
Input your action row:0
Input your action col:0
-------------
| o | o | x |
-------------
|   | x |   |
-------------
| o |   | x |
```

```
        -------------
        -------------
        | o | o | x |
        -------------
        |   | x | x |
        -------------
        | o |   | x |
        -------------
        computer wins!


    # play 1 with human
    p1 = Player("computer", exp_rate=0)
    p1.loadPolicy("/content/drive/MyDrive/policy_p1")

    p2 = HumanPlayer("human")

    st = State(p1, p2)
    st.play2()


     ⍈  -------------
        |   |   |   |
        -------------
        |   | x |   |
        -------------
        |   |   |   |
        -------------
        Input your action row:0
        Input your action col:0
        -------------
        | o |   |   |
        -------------
        |   | x |   |
        -------------
        |   |   |   |
        -------------
        -------------
        | o | x |   |
        -------------
        |   | x |   |
        -------------
        |   |   |   |
        -------------
        Input your action row:2
        Input your action col:1
        -------------
        | o | x |   |
        -------------
        |   | x |   |
        -------------
        |   | o |   |
        -------------
        -------------
        | o | x |   |
        -------------
        | x | x |   |
        -------------
        |   | o |   |
```

```
-------------
Input your action row:1
Input your action col:2
-------------
| o | x |   |
-------------
| x | x | o |
-------------
|   | o |   |
-------------
-------------
| o | x | x |
-------------
| x | x | o |
-------------
|   | o |   |
-------------
Input your action row:2
Input your action col:0
-------------
```

```
# play 1 with human
p1 = Player("computer", exp_rate=0)
p1.loadPolicy("/content/drive/MyDrive/policy_p1")

p2 = HumanPlayer("human")

st = State(p1, p2)
st.play2()
```

```
-------------
|   |   |   |
-------------
|   | x |   |
-------------
|   |   |   |
-------------
Input your action row:2
Input your action col:0
-------------
|   |   |   |
-------------
|   | x |   |
-------------
| o |   |   |
-------------
-------------
|   |   |   |
-------------
| x | x |   |
-------------
| o |   |   |
-------------
Input your action row:1
Input your action col:2
-------------
|   |   |   |
-------------
```

```
| x | x | o |
-------------
| o |   |   |
-------------
-------------
|   |   |   |
-------------
| x | x | o |
-------------
| o | x |   |
-------------
Input your action row:2
Input your action col:2
-------------
|   |   |   |
-------------
| x | x | o |
-------------
| o | x | o |
-------------
-------------
|   | x |   |
-------------
| x | x | o |
-------------
| o | x | o |
-------------
computer wins!
```

✓ 1m 6s completed at 4:39 PM