CODE::

```
# import random
import copy


class TicTacToeState:
    def __init__(self):
        # Init empty board
        self.emptyBoard()
        self.choice = ()
        self.turn = 0

    def emptyBoard(self):
        self.board = [[-1 for _ in range(3)] for _ in range(3)];

    # Str repr of positions
    pos2strMap = {-1: '   ',  # Empty
                  0: ' X ',  # Player 0 (human)
                  1: ' O '}  # Player 1 (computer)

    def pos2str(n):
        return TicTacToeState.pos2strMap[n]

    def printBoard(self):
        print("-----------")
        for row in self.board:
            print("{0}|{1}|{2}".format(TicTacToeState.pos2str(row[0]),
                                       TicTacToeState.pos2str(row[1]),
                                       TicTacToeState.pos2str(row[2])))
            print("-----------")

    def hasEnded(self):
        return any([self.playerWinner(), self.computerWinner(), self.noMoreMoves()])

    def isEmpty(self):
        return all(-1 in l for l in self.board)

    def isDraw(self):
        return not self.playerWinner() and not self.computerWinner() and self.noMoreMoves()

    def noMoreMoves(self):
        return not any(-1 in l for l in self.board)

    def isWinner(self, who):
        # Check horizontals
        for row in range(3):
            if all([x == who for x in self.board[row]]): return True
        # Check verticals
        for col in range(3):
            if all([x == who for x in [self.board[0][col], self.board[1][col], self.board[2][col]]]): return
True
        # Check diagonals
```

```
    if all(self.board[i][i] == who for i in range(3)) or all(
        self.board[i][2 - i] == who for i in range(3)): return True
    return False

def playerWinner(self):
    return self.isWinner(0)

def computerWinner(self):
    return self.isWinner(1)

def placeMove(self, who, row, col):
    self.verifyValidMove(row, col)
    if self.board[row][col] == -1:
        self.board[row][col] = who
        self.turn = 1 - self.turn
    else:
        raise RuntimeError('Place {0},{1} occupied by {2}'.format(row, col, self.board[row][col]))

def verifyValidMove(self, row, col):
    if not row in range(3) or not col in range(3):
        raise RuntimeError('Invalid position')

def placeMovePlayer(self, row, col):
    assert (self.turn == 0)
    self.placeMove(0, row, col)
    self.turn = 1

def placeMoveComputer(self, row, col):
    assert (self.turn == 1)
    self.placeMove(1, row, col)
    self.turn = 0

def gameScore(self):
    if self.playerWinner():
        return -1
    elif self.computerWinner():
        return 1
    else:
        return 0

def availableMoves(self):
    # Returns a list of tuples
    ret = []
    for row in range(3):
        for col in range(3):
            if self.board[row][col] == -1:
                ret.append((row, col))
    return ret

def computerMove(self):
    TicTacToeState.computeNextMoveAt(self, -999999, 999999)
    self.placeMoveComputer(self.choice[0], self.choice[1])
```

```
    def computeNextMoveAt(current_state, alpha, beta):
        # Terminal node
        if current_state.hasEnded():
            return current_state.gameScore()

        moves = []
        scores = []

        # Fill scores and moves, recurseively using minmax
        if current_state.turn == 1:
            move_score = -999999
            for move in current_state.availableMoves():
                next_state = copy.deepcopy(current_state)
                next_state.placeMove(next_state.turn, move[0], move[1])
                move_score = max(move_score, TicTacToeState.computeNextMoveAt(next_state, alpha,
beta))
                moves.append(move)
                scores.append(move_score)
                alpha = max(alpha, move_score)
                if beta <= alpha:
                    break
            current_state.choice = moves[scores.index(max(scores))]
            return move_score

        if current_state.turn == 0:
            move_score = 999999
            for move in current_state.availableMoves():
                next_state = copy.deepcopy(current_state)
                next_state.placeMove(next_state.turn, move[0], move[1])
                move_score = min(move_score, TicTacToeState.computeNextMoveAt(next_state, alpha,
beta))
                moves.append(move)
                scores.append(move_score)
                beta = min(beta, move_score)
                if beta <= alpha:
                    break
            current_state.choice = moves[scores.index(max(scores))]
            return move_score


class TicTacToe:

    def __init__(self):
        # Init empty board
        # int player_score,draw_score,computer_score
        self.state = TicTacToeState()
        self.player_score = 0
        self.draw_score = 0
        self.computer_score = 0

    def startGame(self):
```

```
    self.state.emptyBoard()
    self.state.printBoard()
    self.state.turn = 0
    while not self.state.hasEnded():
        self.playerInput()
        self.state.printBoard()
        if self.checkWinner():     return
        print("Computer's move...")
        self.state.computerMove()
        self.state.printBoard()
        if self.checkWinner():     return

def checkWinner(self):
    if self.state.playerWinner():
        self.player_score += 1
        print("Player wins")
        self.score()
        return True
    elif self.state.computerWinner():
        self.computer_score += 1
        print("Computer wins")
        self.score()
        return True
    elif self.state.isDraw():
        self.draw_score += 1
        print("Draw")
        self.score()
        return True
    return False

def playerInput(self):
    while True:
        inp = int(input("Enter position (1-9 according to dial pad): "))
        if inp < 4:
            inp += 6
        elif inp > 6:
            inp -= 6
        x = int((inp - 1) / 3)
        y = int((inp - 1) % 3)
        # print(x,y)
        try:
            if 1 > inp < 9:
                print("enter input in range")
                continue
            self.state.placeMovePlayer(x, y)
            return
        except (RuntimeError, ValueError) as e:
            print(e)

def score(self):
    print(str(self.player_score) + "\t" + str(self.draw_score) + "\t" + str(self.computer_score))
    print("Player\t=\tcomputer")
```

```
t = TicTacToe()
while (1):
    t.startGame()
    desision = input("Give any input Except 'N' to continue... ")
    if (desision == 'N'):
        break
```

OUTPUT:::