Reinforcement Learning Labsheet 2 - Tic Tac Toe Game

Name: Girish Sundararajan Roll: AM.EN.U4AIE22044

Code:

```
import random

class TTT:
    def __init__(self, auto=None):
        self.grid = [[' ' for _ in range(3)] for _ in range(3)]
        self.current_player = 'X'
        self.game_status = {"ongoing": False, "winner": None, "draw":
False}
        self.auto = auto

    def print_grid(self):
        print("+---+---+---+")
        for row in self.grid:
            print("| {} | {} | {} |".format(*row))
            print("+---+---+---+")

    def start_game(self):
        self.print_grid()
        self.game_status["ongoing"] = True
        if self.auto is None:
            while self.game_status["ongoing"]:
                self.player_move()
                self.check_winner()
                self.print_grid()
                if self.game_status["ongoing"]:
                    self.switch_player()
        else:
            self.print_grid()
```

```python
            while self.game_status["ongoing"]:
                valid = self.give_valid_moves()
                move = random.choice(valid)
                self.grid[move[0]][move[1]] = self.current_player
                self.check_winner()
                self.print_grid()
                if self.game_status["ongoing"]:
                    self.switch_player()
                self.player_move()
                self.check_winner()
                self.print_grid()
                if self.game_status["ongoing"]:
                    self.switch_player()

    def player_move(self):
        while True:
            try:
                move = int(input(f"Player {self.current_player}, enter
your move (1-9): ")) -1
                if move < 0 or move > 8:
                    print("Invalid move. Please enter a number between 1
and 9.")
                    continue
                row, col = move // 3, move % 3
                if self.grid[row][col] != ' ':
                    print("Cell already taken. Choose another one.")
                    continue
                self.grid[row][col] = self.current_player
                break
            except ValueError:
                print("Invalid input. Please enter a number between 1 and
9.")
            except Exception as e:
                print(f"An error occurred: {e}")
                continue




    def give_valid_moves(self):
        valid = []
        for i in range(3):
            for j in range(3):
                if self.grid[i][j] == ' ':
                    valid.append((i, j))
        return valid

    def switch_player(self):
        self.current_player = 'O' if self.current_player == 'X' else 'X'
    def check_winner(self):
        lines = self.grid + [[self.grid[j][i] for j in range(3)] for i in range(3)] + [[self.grid[i][i] for i in range(3)], [self.grid[i][2-i] for i in range(3)]]
        win_cond = [["X", "X", "X"], ["O", "O", "O"]]
        for line in win_cond:
            if line in lines:
                self.game_status["ongoing"] = False
                self.game_status["winner"] = line[0]
                print(f"\n\nPlayer {line[0]} wins!\n\n")
                return True
        return self.check_draw()
    def check_draw(self):
        if all(cell != ' ' for row in self.grid for cell in row):
            self.game_status["ongoing"] = False
            self.game_status["draw"] = True
            print("\n\n\nIt's a draw!\n\n")
            return True
        return False

if __name__ == "__main__":
    game = TTT()
    game.start_game()
```

Question and Answers

1. I implemented the automated player by collecting all empty cells and then choosing one at random. This Computer agent is not intelligent as it randomly selects the cells instead of trying to win or block the other player.

2. The main challenge was covering all rows, columns, and diagonals. I solved this by combining rows, building columns with loops, and explicitly adding the two diagonals. This way all 8 winning lines are checked.

3. I used a function that scans the grid and collects all empty spots. The automated player then picks one randomly from this list, so it never chooses an invalid move.

4. a. In one game, I played X and the bot was O. My moves were 1, 2, 3 while the bot picked 5 and 9.
b. Final board:

```
X | X | X
  | O |
  |   | O
```

X won with the top row.
c. Yes, I used random strategy. It made the bot weak since it didn't block me, and I won easily.
d. With two human players, moves were 1, 5, 9, 2, 3, 8, 7.
e. Final board:

```
X | O | X
  | O |
X | O | X
```

X won with a diagonal.

5. I handled invalid inputs with checks. If the number is not 1–9, or if the cell is taken, or if input is not a number, the program prints an error and asks again. For example, entering 12 or choosing an occupied cell is rejected.

6. The flaw is that the bot only plays randomly. A better strategy would be to use Minimax so it can block the opponent or win when possible.

7. An improvement would be making the bot smarter. This would make the game more challenging and fun compared to the current random play.

8. The game mostly worked fine, but one issue was that sometimes the bot wins but the program still asks me for a move before ending. I debugged this by checking game status after every move.