Cairo University
Faculty of Engineering
Computer Department
4th year

Backgammon

Submitted to Eng. Yahia Zakaria

Student Name	**Ayman Shawky Shehata Mohammed Azzam**
Student Section	**1**
Student Bench Number	**11**
Problem Number	**5**

Code Explanation

1. **Board Representation**: The two players are defined as player 1 (white) and player -1 (red). The board has 29 positions as following:
    a. Position 0 isn't used.
    b. Positions 1-24 Contain the number of checkers in that position.
    c. The position 25 contains the number of killed checkers for player 1.
    d. The position 26 contains the number of killed checkers for player -1.
    e. The position 27 contains the number of checkers have been got it out for player 1
    f. The position 28 contains the number of checkers have been got it out for player -1
    g. Positions for player 1 are positive values while positions for player -1 are negative values, for examples:
        i. board[23] = 3, means that player 1 has 3 checkers on the 23rd position of the board.
        ii. board[21] = -10, means that player -1 has 10 checkers on the 21st position of the board.
        iii. board[25] = 1, means that player 1 has 1 killed checker.
        iv. board[28] = -2, means that player -1 has got out two checkers.

2. **Backgammon.py**
    a. It initializes the board then randomizes on which player to start. You can control the maximum depth of the expectiminimax tree in this file using the variable max_depth.
    b. Then it starts the game between two players using the same agent and stops the game when any of them wins.
3. **Agent.py**
    a. It gets the available moves then choose the best one between them.
    b. It evaluates between them using the evaluation function explained in this Article
    c. It uses the expectiminimax tree as explained in the reference
    d. Inputs:
        i. Board: numpy array of size 29
        ii. Dice: numpy array of size 2
        iii. Player: integer (1 or -1)
        iv. Max_depth: integer
    e. Outputs:
        i. Board: numpy array of size 29 (the new board after the agent's play)

How to Run the Code

1. Dependencies: you need only numpy
2. Use the following command to run it: "python backgammon.py"

Problem Answers

Increasing the Search Depth	Evaluation Function
Increasing the Search Depth should improve the agent expectation but at the same time it costs much complexity so maximum depth you can use in normal computer is 4	Improving the Evaluation function will definitely improve the agent results but at the same time it's hard to measure the performance of the agent but you can improve between two agents to know which one is better.

How I Wrote the Code

I searched on the internet how the other people handled this problem then I chose reasonable solutions for the backgammon and I implemented the solution(I used some parts as in the search for example the part of generation all possible moves). For the agent part I implemented the expectiminimax by myself and I chose the evaluation function from an Article and I implemented it with a small edit.

Conclusion

1. It's impossible to implement optimal agent that always win for the backgammon because it depends on the dice probability
    a. Example on that when you run the agent sometimes player 1 will win and other times player -1 will win.
2. The agent depends much on the evaluation function so it's the most important design element in this problem.