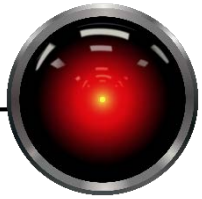


Assignment 3

Part I: Questions (20 points each)

1. Backgammon rolls are between 2 and 12. Find the probability for each pair of numbers. That is without distinguishing between (5,6) and (6,5).

   The outcome of each die is independent of the other. Assuming fair dice, the probability of each die roll is 1/6, and the probability of a specific number on each die is the product of the the two dice: 1/6*1/6=1/36. For a pair of dice to roll two numbers, in cases where the numbers are different, it can be done in one of two ways. As an example with a red die and a blue die, one can roll the pair (2,3) with (red=2, blue=3) or (red=3, blue=2). As a consequence, the probability of rolling (2,3) is 2/36 (1/36 + 1/36).

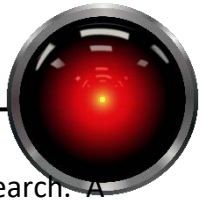
   $$P\Big(\big(X=x,Y=y\big) \text{ or } \big(X=y,Y=x\big)\Big)=\begin{cases}\frac{1}{36} & x=y\\ \frac{2}{36} & x\neq y\end{cases}$$

   The question does not ask for the probability of rolling a specific number as the sum, which is important for games like craps, but not as much for backgammon. If you answered the sum problem and did so correctly, you will have answered the problem "mostly right." Example: A 9 can be rolled with a 6 and 3 or 7 and 2. Hence, the probability of rolling a 9 is 2/36+2/36=4/36.

2. Explain in your own words the role of chance nodes in stochastic minimax search and how they relate to classic minimax search.

   The chance nodes allow for nondeterministic actions whose results the agent cannot control such as a die roll. Probabilities are assigned to each action according to the likelihood of events occurring. It's role in the stochastic minimax search is very similar to that of classic minimax search. In a traditional minimax search, we seek to maximize or minimize a ply depending upon the player's goals (maximize/minimize utility). However, in stochastic plies, there is no control and instead we look at the expected outcome. That is, we conduct search across each possible outcome and weight the results by the probability of their occurrence.

3. The CheckerBoard class in the programming assignment below cannot detect stalemates that occur when the board is in the same configuration three different times. How could one modify the class to detect this efficiently? (No implementation is required, just well laid out plan that could be implemented by any skilled computer scientist.)



This could be solved very similarly to the explored set class in graph search. A data structure could store each state when a move occurs and augment the state with an integer count. Each time a move is made, the set of states is searched (hopefully efficiently using a hash table). If the state has never been seen, it is added to the data structure with a count of 1. If it has been seen before, the counter is incremented and a stale mate is declared if it reaches 3.