Jinqi Cheng Assignment 4 Part 1

Problem 1

Answer:

```
2 :  0.0625
3 :  0.125
4 :  0.1875
5 :  0.25
6 :  0.1875
7 :  0.125
8 :  0.0625
```

Work:

```
count = 0 #total number of solutions
dic = dict() #key: 2 to 8, value: number of key appearances

#dice 1
for i in range(1,5):

    #dice2
    for j in range(1,5):
        s = i+j
        if s in dic:
            dic[s] += 1
        else:
            dic[s] = 1
        count+=1

for key,value in dic.items():
    print(key,": ",value/count)
```

Problem 2

$$\begin{aligned} utility(\Gamma) &= P(A)\times 10 + P(B)\times 15 + P(C)\times 29 \\ &= 0.25\times 10 + 0.50\times 15 + 0.25\times 29 \\ &= 17.25 \end{aligned}$$

$$\begin{aligned} utility(\Psi) &= P(D)\times 14 + P(E)\times 18 \\ &= 0.45\times 14 + 0.55\times 18 \\ &= 16.2 \end{aligned}$$

$\because utility(\Gamma) > utility(\Psi)$
$\therefore$ the player should choose Γ

Problem 3

We can create a function called isStalemate(). This function returns a boolean value and take two parameters:
self - the checkerboard
dic - a dictionary that stores number of times that each board state appears
dic - key: checkerboard repr, value: number of appearances
This function should be called after each move() is performed
The following is a piece of pseudocode.

```
def isStalemate(self, dic):
   #currentboard is a string
   currboard = self.__repr__()

   if currboard in dic:
      dic[currboard] += 1
   else:
      dic[currboard] = 1

   if dic[currboard] >= 3:
      return true
   return false
```

We can optimize this solution by clear the dic after a piece been captured, since any board state after that cannot be the same as the state before capturing the piece. This saves the memory consumption by avoiding the hashmap been too big. We can perform this inside move() method:

```
def move(self, move, validate=[], verbose=False):
    ...
    if a piece is captured:
        dic.clear()
    ...
```