Writing this program to beat this AI could be seen as a daunting task because there are no greedy solutions. In addition, the website does all the calculation on the server-side, so we do not know how many times it has trained, or how it calculates its answer. The AI very quickly learns from repeated patterns, even patterns of varying length. It also factors in various human biases, like wanting to even out the number of each choice to make a uniform distribution, or not expecting a large amount of one choice in a row. The AI also has learned to bait people into victories by relying on predicted switches. That is, it repeatedly chooses a singular option, and when it believes that the human will change to counter, it counters the counter. People do this too, but with less conviction, meaning they try to do it too quickly, and are unwilling to choose a singular option too many times.

Considering these factors rely on human error, the best solution would be to compile the programs responses to patterns, and decide based on the probability that it chose an answer in the past. Again humans can discern these patterns too, and can use them successfully against the AI, but in 100 trials, fatigue kicks in and we do not see the same level of effort in evaluating the best solution in each trial, leading to a suboptimal performance. In the 100 trials I performed randomly, I had a win-loss ratio of .77. Towards the later trials I found myself losing more often, even though I had built a small lead in the first 30 trials. This could be attributed to fatigue in recognizing patterns, or that the AI was learning from my pattern.

In the end I relied on a simple Markov Chain to help determine which solution to pick. We tally up every consecutive subset of 3 in the history, meaning ABCD makes ABC and BCD. Then keep track of the last 2 answers the computer provided and use that to determine what the next answer should be. We calculate how many times the possible answer the AI will give have occurred, given the last two answers, and its previous history. Then we determine which is most likely, by subtracting the counts of the other answers, and normalizing. This gives us a distribution, and we then generate a random number and choose an answer according to the distribution. At the end the user provides us with what the AI answered, and then we update the history and the last 2.

In this solution we both keep track of old answers, and have some element of risk taking, given by the random number. This solution had a win/loss ratio of 1.31, much better than a human in 100 trials. The reason we save two answers in our chain is because, with only one answer in the chain we often overreact to the AI repeatedly choosing one solution, forcing us into predicable switches, which can cause more losses. One possible improvement to the algorithm would be possibly trying to keep patterns of 4 answers in the chain, and remembering 3 previous answers. This would allow us to be more precise with our probabilities and pattern sensing, but may make us too slow to react on repeated choices. The length of the pattern is hard to determine, without trial and error.

Code:

```
from __future__ import division
from math import sqrt
import random as rnd

def checkGame(a, b):
    if a == '0' and b == '1' or a == '1' and b == '2' or a == '2' and b == '0':
        return -1
    elif a == b:
        return 0
```

```
    else:
        return 1


#baseline 3 to avoid skewed distribution from history
#(if we start at 0, 1 )
RPS_count = { '000' : 3, '001' : 3, '002' : 3, '010' : 3, '011' : 3, '012' : 3, '020'
: 3, '021' : 3, '022' : 3, '100' : 3, '101' : 3, '102' : 3, '110' : 3, '111' : 3,
'112' : 3, '120' : 3, '121' : 3, '122' : 3, '200' : 3, '201' : 3, '202' : 3, '210' :
3, '211' : 3, '212' : 3, '220' : 3, '221' : 3, '222' : 3 }

RPS_disp  = {'0' : 'rock', '1' : 'paper', '2' : 'scissor'}

wins, ties, losses = 0,0,0
last2 = '33'

#get history of AI answers stored as R,P,S in the first line of a file
with open('history.txt', 'r') as history:
    for line in history:
        temp = line
        break

temp = temp.replace('R', '0')
temp = temp.replace('P', '1')
temp = temp.replace('S', '2')

#print(temp)

for x in range(0, len(temp)-2):
    RPS_count[temp[x:x+3]] += 1

#print(RPS_count['000'])

while(1):
    if(last2[0] == '3'):
        y = str( rnd.randint(0,2) )
    else:
        #counts of all possible answers
        r_count = RPS_count[last2 + '0']
        p_count = RPS_count[last2 + '1']
        s_count = RPS_count[last2 + '2']

        tot_count = r_count + p_count + s_count

        q_dist = [ r_count/tot_count, p_count/tot_count, 1- (r_count/tot_count) -
(p_count/tot_count) ]

        #getting distribution and normalizing
        result = [ max(q_dist[2]-q_dist[1],0), max(q_dist[0]-q_dist[2],0),
max(q_dist[1]-q_dist[0],0) ]
        resultnorm = sqrt(result[0]*result[0] + result[1]*result[1] +
result[2]*result[2])
        result = [result[0]/resultnorm, result[1]/resultnorm, 1 -
result[0]/resultnorm - result[1]/resultnorm]
```

```
        #picking random nuimber from distribution
        y = rnd.uniform(0,1)

        if y <= result[0]:
            y = '0'
        elif y <= result[0] + result[1]:
            y = '1'
        else:
            y = '2'

    print("Enter " + str(RPS_disp[y]))
    print("What did the computer enter? \n")

    roll = input('Please type r,p,s, or q\n')

    while(roll not in ['r', 'p', 's', 'q']):
        roll = input("Look: you've got to type r,p,s, or q\n")

    #print(roll == 'r')
    #print(roll)
    if roll == 'r':
        x = '0'
    elif roll == 'p':
        x = '1'
    elif roll == 's':
        x = '2'
    elif roll == 'q':
        break

    #update history
    if last2[0] != '3':
        RPS_count[last2+x] += 1

    last2 = last2[1] + x

    print(last2)

    print ('I played: ' + str(RPS_disp[y]) + '\nComputer played:   ' +
str(RPS_disp[x]) + '\nGAME RESULT (-1 is a loss for me): ' + str(checkGame(y,x)))

    if checkGame(y,x) == -1:
        losses += 1
    elif checkGame(y,x) == 0:
        ties   += 1
    elif checkGame(y,x) == 1:
        wins   += 1

    print ('Wins:', wins, 'Losses:', losses, 'Ties:', ties)
    print("\n")
```