ANGLICAN INTERNATIONAL SCHOOL JERUSALEM

Tarneeb: An Investigation of a Middle Eastern Card Game from a Mathematical Perspective

IB Mathematics HL Internal Assessment

By Rami Manna

May 2014

CANDIDATE NUMBER: 1043-0009

TABLE OF CONTENTS

TARNEEB: AN INVESTIGATION OF A MIDDLE EASTERN CARD GAME FROM A MATHEMATICAL PERSPECTIVE

INTRODUCTION

In Tarneeb, two teams of two players sit across from each other. Thirteen cards are dealt to each player, which means a whole deck without jokers is used. A round of bidding occurs before each game, in which players announce the number of tricks they think their team can win for their preferred trump suit. The highest bidder chooses the trump (or 'tarneeb') and the game begins. Consisting of 13 rounds, the bidder's team must make at least the number of tricks declared in the bid in order to win. In each round, all players must play the same suit as the first card placed; the highest card played wins the round. In the case that a player does not have the suit in play, they may play any other card, and have the choice of placing a tarneeb, in which case the tarneeb will overpower even the highest card of the suit in play.

Tarneeb is one of the most popular card games of Arab culture. It is played by many of my relatives, and has probably been passed down many generations in my family. My dad first taught me the game when I was 11, and already fascinated with card games. I fell in love; Tarneeb quickly became the only game I played for a long time. I taught the game to the closest of my friends, my favorite teacher and eventually my whole Math HL class. It was at this point that I realized that people with more mathematical minds seemed to play better and win more often than those less mathematically gifted. I became motivated to find an explanation for this phenomenon.

Is there a mathematical aspect of the game? If so, can the mathematical aspects of Tarneeb be distinguished from the non-mathematical aspects?

To explore the mathematical aspects of Tarneeb, I intend to start by interviewing expert players. This will broaden my knowledge of Tarneeb strategies, providing me with a better perspective of the game. I will implement the new techniques the experts present into my set of playing techniques. If given the proper instructions and a set of in-game techniques, a computer could play a Tarneeb game. Collecting data from games played solely by humans would be a messy and long process, not only because each game takes about 5 minutes, each player would also have to write down their cards at the beginning of each game and calculations concerning the length and strength of their Tarneeb would also be necessary. There are computer applications both for smartphones and websites where human players can interact with computerized players in a Tarneeb game.[1][2] Computers can work at much faster speeds than humans, which makes writing a program a practical means of attaining a large amount of data on Tarneeb games.

Originally, the plan was to create a user interface (a computer game) where I would get humans to play the game against 3 other players controlled by the computer. This would be more efficient than having a real life game as the computer can log all the data as it goes and the 3 computer controlled players play their cards instantly (only time wasted is on the human player's turn). However, after I began to code this user-

[1] " Tarneeb" - Android App by Emad Jabareen

[2] tarneeb.com

interface, I realized that it would take me a lot of time to do so, and even after I was done, I would need to spend more time having people play the game. So I changed my plan once again, if I could get a computer to play, it could simulate a whole game of Tarneeb on its own, without a single human player. With this method, a computer could play about 200 games and log calculations on each game in a single minute.

After enough trials to reduce randomness, I plan to analyze my data to find the best bidding strategies, by investigating length of suit while controlling strength of card hand at three different levels. Can I build a model for tarneeb bidding strategies?

INTERVIEWS WITH EXPERT PLAYERS

I interviewed four veteran players who have each been playing for over 5 years. I used a semi-structured questionnaire shown in Appendix 1 with the responses from the interviewees. A summary of the strategies mentioned follows.

CALLING STRATEGIES

Many websites deal with strategies but give little advice about opening bids. I was most interested in the views of the expert players on this point. Most of the interviewees did not talk much about choosing which suit to call, which is probably because they've played for so many years that recognizing the best fit suit to be tarneeb becomes instinctive. Instead, the focus was on how much to call, and then, most seemed to put a lot of emphasis on the length of the chosen tarneeb suit. According to some of the interviewees, the **LENGTH** of the tarneeb suit was sufficient as the only

factor to be assessed when making a call. Others claimed that the number of high cards (proportional to **STRENGTH** of hand) needs to be taken into account as well.

INGAME COUNTING STRATEGIES

All interviewees claimed to count tarneeb, keeping track of how many tarneebs have been played, and even which tarneeb cards have been played. In addition to counting tarneeb, all of the experts said they kept track of the high cards of all suits.

IN-GAME PLAYING STRATEGIES

Social Game:

- Intimidate Opponents: Even if your cards aren't good, pretending they are may cause your opponents to scare from calling high.
- Watch Partner's Reaction to Cards

Specific to Caller:

- Always wipe tarneeb out at the beginning
- Quickly empty out your shortest suits from your hand, so that you can tarneeb on them
- If you have a long suit that you don't call on, leave it to the end so that you can "loot" it at the end when everyone is out of tarneeb

General Strategies:

- Always play your highest card if you’re third in a round (unless someone else has already played a higher card than yours)
- If you have the ace of a tarneeb and only low cards, do not play the ace at the beginning of the game
- Never start with a J or Q, unless it's your last card of a suit AND you still have tarneeb
- When starting a round, play the highest card of your suit, if it's the highest still in the game, but your lowest card of your shortest suit otherwise (in the case that you still have tarneeb)

IMPLEMENTING STRAGEGIES

After interviewing several Tarneeb experts, I decided to implement the new techniques they presented into my set of playing techniques. These new playing strategies quickly became a part of my instinctive playing routine. I soon noticed that my playing had become even more based on principles than it was before, which meant that I rarely found myself in indecision when it came to deciding what card to play on a given round. Playing Tarneeb has become a straightforward and almost mechanical process. This made me realize that if given the proper instructions and my set of in-game techniques, a computer could play a Tarneeb game almost identically to me.

PYTHON PROGRAM

I acquired programming skills during the MEET[3] summer program (Middle Eastern Education through Technology), an indirect peace initiative where Israeli students, Jews and Arabs, learn computer science together. One of the programming languages that were taught to the students was Python. This seemed a practical and efficient way of analyzing Tarneeb games.

To simulate a Tarneeb game, I had to write step by step instructions. I incorporated the experts' strategies mentioned above and my own experience of the game into these instructions. A small sample of the 400 line program is shown on the next page; it deals with how the third player in a round should proceed:

[3] meet.mit.edu

```
  def third(self, player):
        #print "third",str(self.currentplay)
        suit_to_play = self.currentplay[(player+2)%4]/13
        if self.have_suit(player,suit_to_play): # if you have suit
            beat = False
            suit_begin_index, suit_end_index = self.suit_begin_end(player,
suit_to_play)
            suit_end_index -= 1
            if self.players[player][suit_end_index]-
self.currentplay[(player+2)%4]>1:  #ERROR list index out of range
                for x in
xrange(self.currentplay[(player+2)%4]+2,suit_end_index+1): #for all cards in
between your card and parters   #ERROR List index out of range
                    if x not in self.played_cards:
                        beat = True            if
self.currentplay[0]>self.currentplay[1] and beat: #partner winning, only beat
partner if gap between your highest card
                if self.playcardifhave(player,
self.players[player][13*suit_to_play+self.suitQ[player][suit_to_play]-1]):
                    return
            else: #partner losing or not worth beating
                for v in xrange(13*suit_to_play,13*suit_to_play+13):
                    if self.playcardifhave(player,v):
                        return
        if self.currentplay[1]>self.currentplay[0] and self.have_suit(player,
self.tarneeb):# partner losing -> tarneeb lowest if can
            for v in xrange(13*self.tarneeb,13*self.tarneeb+13):
                if self.playcardifhave(player,v):
                    return
        if not self.suits_owned_by_player[player][self.tarneeb]:
            for k in
xrange(13*self.longest_suit[player],13*self.longest_suit[player]+max(self.suitQ
[player])):
                if self.playcardifhave(player,k):
                    return
        else: # have tarneeb but partner winning, play lowest card of shortest
suit
            callerSuitQ = self.current_suit_lengths(player)
            callerSuitQ[self.tarneeb] = 99 #Not playing tarneeb just because it
happens to be the shortest suit now.
            for i in xrange(3):
                if callerSuitQ[i] == 0:
                    callerSuitQ[i] = 99
            if min(callerSuitQ) == 99: #only tarneeb left. Play tarneeb.
                if
self.playcardifhave(player,self.players[player][len(self.players[player])-1]):
#PLAY HIGHEST TARNEEB
                    return
            shortestSuit = callerSuitQ.index(min(callerSuitQ))
            for j in range(13*shortestSuit, 13*shortestSuit + 13):
                if self.playcardifhave(player,j):
                    return
        #print "failsafe 3"
        self.playcardifhave(player, self.players[player][0])
    #@returns round_winner
```

The infrastructure for the game also has to be set up; this includes shuffling and dealing cards, setting the tarneeb suit, keeping track of scores, calculating the length

and strength of the tarneeb suit. While counting the length of the tarneeb suit is relatively easy, determining the strength required more input from me.

Points were assigned to every card (see Table 1) unlike bridge where only picture cards and aces are assigned points. The sum of all the points in the tarneeb suit indicates the strength. I have categorized a sum of strengths of -5 to -1 as 'Weak', 0 to 4 as 'Medium Weak', 5 to 9 as 'Medium' and 10 to 14 as 'Strong'.

Table 1: Strength Point System

Card in Tarneeb suit	Points
2	-6
3	-5
4	-4
5	-3
6	-2
7	-1
8	0
9	1
10	2
J	3
Q	4
K	5
A	6

Being a novice at coding, writing a 400 line program on a complex game was a larger undertaking than I had originally imagined. I had to learn OOP (Object Oriented Programming) in order to write the program. Debugging the program also took a lot of time as each time an error is fixed, a new error might arise.

The program generated 400,000 games over the course of 30 hours. Had I used an existing program, such as Emad Jabareen's android app, where one of the players

must be human, it would have taken 3.8 years to generate the same amount of data, without the implementation of the expert strategies from the interviews.

The huge amount of data generated caused technical difficulties as the files were too large to be opened by most programs. Microsoft Excel could open the file, but I could not handle the data as it was in strings as opposed to integers. I had to write another Python program which converted the strings into integers and tallied the scores for games of different lengths and strengths.

This gave us a sufficiently large pool of data that could be mined for the variables of length and strength. Even with 400,000 games, there were only 8 games with length 9 however occurrences of 9 of the same suit is an extreme case (0.0000926 chances). Lengths 7 and 8 are also unlikely with probabilities of less than 1%. In contrast to the computer simulation, where cards are dealt randomly, in real life, shuffling or lack thereof plays a role. Since each round has cards of the same suit, at the end of the game, cards are not in a random order and suits tend to be packed together. It would take a lot of shuffling to create a random order. Some players take advantage of this packing by shuffling minimally and dealing cards 13 at a time. Others, such as interviewee Ahmad, frown on this practice. From my experience, these practices lead to long suits (7 and 8 coming up at a greater rate than the theoretical probability) which make the game more competitive. I have looked at the data for lengths 5-8 with weak, medium and strong tarneeb strengths.

RESULTS

From the expert interviews, the two factors that emerged as influencing the opening bid were length of suit and strength. I have controlled strength and calculated the probabilities of making different calls for increasing length. Probabilities that are greater than 0.5 are highlighted in Table 2. Probability of making a call is calculated by adding the total number of games for which a total score of the call or higher is made (only for games with the specific length and strength under consideration), and dividing that by the total number of games with the specific length and strength under consideration. Table 2 shows the probabilities of making calls for increasing length over a range of controlled strengths:

Table 2: Probabilities of making calls for increasing lengths for different strengths

Weak:

Length / Call	5	6	7	8
7	0.533	0.7	0.836	0.952
8	0.298	0.463	0.592	0.838
9	0.123	0.226	0.338	0.638
10	0.0352	0.0679	0.134	0.314
11	0.00729	0.0129	0.0378	0.124
12	0.000782	0.000789	0.00391	0.0191
13	0	0	0	0

Medium Weak:

Length / Call	5	6	7	8
7	0.626418	0.770906	0.914683	0.985612
8	0.372577	0.533101	0.71131	0.906475
9	0.165183	0.272213	0.401786	0.705036
10	0.053841	0.096908	0.178571	0.338129
11	0.010625	0.019164	0.050595	0.093525
12	0.001292	0.002178	0.005952	0
13	0	0	0	0

Medium:

Length / Call	5	6	7	8
7	0.728	0.859	0.951	1
8	0.47	0.643	0.842	0.923
9	0.229	0.355	0.554	0.769
10	0.0788	0.135	0.25	0.4
11	0.0174	0.0284	0.0663	0.0308
12	0.00263	0.00222	0.00195	0
13	0.000146	0	0	0

Strong:

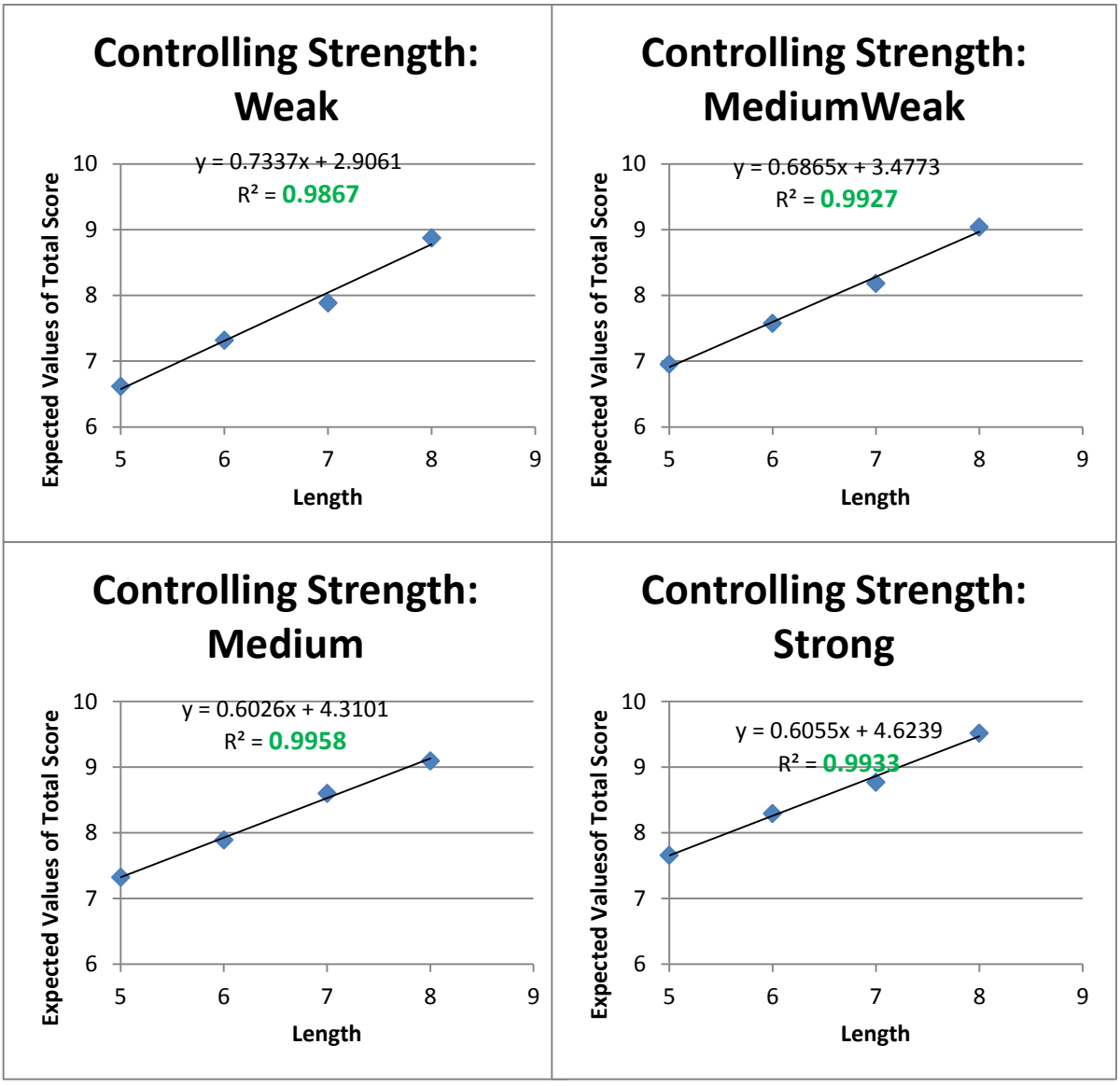
Length / Call	5	6	7	8
7	0.8	0.932	0.979	1
8	0.553	0.729	0.839	1
9	0.299	0.432	0.565	0.957
10	0.0988	0.176	0.311	0.522
11	0.0202	0.0362	0.0881	0.261
12	0.00282	0.00566	0.00518	0
13	0	0.00226	0	0

Although calling with high probabilities of winning would be ideal (upper data points of the blue shaded regions of Table 2), it is important to remember that the determination of the call involves a bidding process. This means that one should call the highest number of tricks for which they will still have a better chance of winning than losing (above 0.5 chance of success). This is a precautionary strategy as it minimizes the chance that the opponent has a higher probability of winning than you if they decide to step up your call. At the same time, if the opponent decides not to call higher than you, you will still have a higher chance of winning. The values with greater chances of success than loss are shaded blue in the Table 2. The call that would be made according to this strategy is the greatest possible call that remains in the blue shaded region; these data points are outlined with black lines.

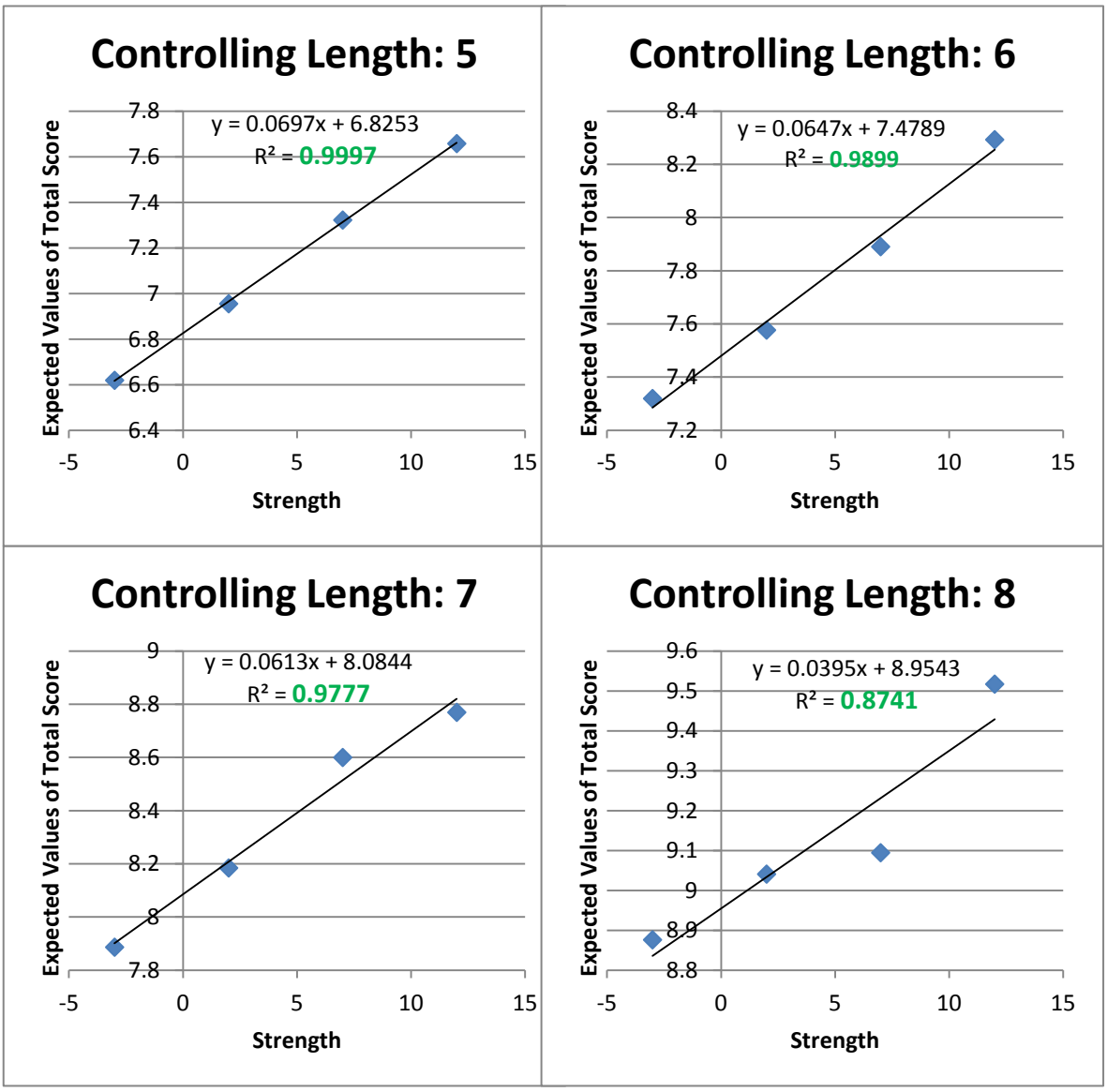
These tables can help players make their calls, and their format is good because they are easy to refer to if faced with indecision whilst making a call in a real Tarneeb game. However, my goal is to eventually create a model, i.e. a single equation that could fit on the palm of one's' hand, only requiring a single calculation to decide a call with strength and length as parameters. In order to know whether or not this is achievable, I must first see if my data shows strong positive correlations between length and total score, as well as strength and total score. From this point on, I will be using the expected values of the total score rather than the probability of making the call. This will give a better idea of what total score will be made in practice, and result in a model that allows one to expect the final result of a game before playing it.

Graphs 1 and 2 show the correlations for the expected values of total scores for controlling strength and length respectively, whilst varying the other each time.

Graph 1: Expected Values of Total Scores for increasing lengths whilst controlling strength at different categories



Graph 2: Expected Values of Total Scores for increasing strengths whilst controlling length at different values



The R^2 values (which are coefficients of determination that explain variance in Y from in X) are labeled in green, and all indicate exceptionally strong correlations. The only apparent drop in correlation occurs for "Controlling Length: 8" in Graph 2, which can be explained as a result of insufficient data for the highly unlikely combination of length 8 with a medium or strong tarneeb suit. Nonetheless, we have shown that whilst

the other factor is controlled, length and strength both have almost direct effects on total score. This means that with a combination of the variables of strength and length, we can achieve a reliable model for estimating total score with a very low uncertainty/error for the given assumptions (for my computer simulation).

BUILDING A MODEL BASED ON RESULTS

The results have shown exceptionally strong positive linear correlations for total score for increasing length and strength (whilst controlling the other variable each time). Having established that there is a linear correlation between length and score and a linear correlation between strength and score, I will least squares to build a model that expresses (expected) total score as a linear combination of strength and length. The following process is with the goal of projecting the data onto the following formula: CL + DS +EP = 1, where L = length, S = strength, P = points (total score) and C, D & E are constants (which I will calculate).

Using the data points to create Matrix A (Columns representing length **L**, strength **S** and Points **P** respectively), we can express a perfect correlation with the following formula: $AX = B$ where

$$A = \begin{bmatrix} 5 & -3 & 6.6194 \\ 6 & -3 & 7.3196 \\ 7 & -3 & 7.8866 \\ 8 & -3 & 8.8762 \\ 5 & 2 & 6.9553 \\ 6 & 2 & 7.5769 \\ 7 & 2 & 8.1845 \\ 8 & 2 & 9.0410 \\ 5 & 7 & 7.3225 \\ 6 & 7 & 7.8906 \\ 7 & 7 & 8.6000 \\ 8 & 7 & 9.0946 \\ 5 & 12 & 7.6579 \\ 6 & 12 & 8.2930 \\ 7 & 12 & 8.7698 \\ 8 & 12 & 9.5172 \end{bmatrix}, X = \begin{bmatrix} C \\ D \\ E \end{bmatrix} \text{ and } B = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

However, there is no one set of values of constants C, D and E that would satisfy this equation. Therefore, we must find the values of these constants that are, overall, the best fit for the data points.

This can be done by projecting B onto A, which will require the projection vector P, acquired using the formula $P = A(A^TA)^{-1}A^T$ where A^T is A transposed.

```
P =

  Columns 1 through 8

    0.2655    0.2044    0.1220    0.1072    0.1982    0.1245    0.0486    0.0125
    0.2044    0.1848    0.1548    0.1578    0.1330    0.1072    0.0804    0.0730
    0.1220    0.1548    0.1910    0.2162    0.0471    0.0820    0.1172    0.1459
    0.1072    0.1578    0.2162    0.2500    0.0267    0.0819    0.1380    0.1795
    0.1982    0.1330    0.0471    0.0267    0.1801    0.1027    0.0232   -0.0179
    0.1245    0.1072    0.0820    0.0819    0.1027    0.0807    0.0579    0.0499
    0.0486    0.0804    0.1172    0.1380    0.0232    0.0579    0.0932    0.1190
    0.0125    0.0730    0.1459    0.1795   -0.0179    0.0499    0.1190    0.1650
    0.1358    0.0640   -0.0285   -0.0556    0.1668    0.0828   -0.0034   -0.0511
    0.0536    0.0340    0.0077    0.0028    0.0812    0.0576    0.0333    0.0217
   -0.0060    0.0152    0.0402    0.0529    0.0174    0.0409    0.0647    0.0813
   -0.1000   -0.0205    0.0784    0.1155   -0.0796    0.0113    0.1043    0.1609
    0.0685   -0.0075   -0.1034   -0.1361    0.1487    0.0610   -0.0288   -0.0814
   -0.0031   -0.0322   -0.0689   -0.0816    0.0734    0.0398    0.0054   -0.0149
   -0.0999   -0.0693   -0.0303   -0.0179   -0.0264    0.0092    0.0456    0.0664
   -0.1534   -0.0852    0.0012    0.0300   -0.0843   -0.0053    0.0756    0.1225

  Columns 9 through 16

    0.1358    0.0536   -0.0060   -0.1000    0.0685   -0.0031   -0.0999   -0.1534
    0.0640    0.0340    0.0152   -0.0205   -0.0075   -0.0322   -0.0693   -0.0852
   -0.0285    0.0077    0.0402    0.0784   -0.1034   -0.0689   -0.0303    0.0012
   -0.0556    0.0028    0.0529    0.1155   -0.1361   -0.0816   -0.0179    0.0300
    0.1668    0.0812    0.0174   -0.0796    0.1487    0.0734   -0.0264   -0.0843
    0.0828    0.0576    0.0409    0.0113    0.0610    0.0398    0.0092   -0.0053
   -0.0034    0.0333    0.0647    0.1043   -0.0288    0.0054    0.0456    0.0756
   -0.0511    0.0217    0.0813    0.1609   -0.0814   -0.0149    0.0664    0.1225
    0.2027    0.1104    0.0399   -0.0638    0.2336    0.1516    0.0452   -0.0194
    0.1104    0.0841    0.0649    0.0349    0.1379    0.1150    0.0841    0.0669
    0.0399    0.0649    0.0858    0.1130    0.0633    0.0864    0.1141    0.1339
   -0.0638    0.0349    0.1130    0.2225   -0.0434    0.0456    0.1576    0.2302
    0.2336    0.1379    0.0633   -0.0434    0.3138    0.2281    0.1187    0.0498
    0.1516    0.1150    0.0864    0.0456    0.2281    0.1953    0.1534    0.1270
    0.0452    0.0841    0.1141    0.1576    0.1187    0.1534    0.1981    0.2257
   -0.0194    0.0669    0.1339    0.2302    0.0498    0.1270    0.2257    0.2875
```

As there is not a set of values of C, D and E that satisfy the equation $AX = B$, I will replace B with PB, i.e. the projection of B onto the column space of A (the space spanned by the column vectors in A). While the vector of B does not lie on the same plane as A, vector PB is projected onto the same plane as A. This means that for $AX = PB$, I will be able to solve for X, i.e. find a single solution for constants C, D and E.

$$\text{PB} = \begin{bmatrix} 0.9785 \\ 0.9938 \\ 0.9726 \\ 1.0672 \\ 0.9904 \\ 0.9842 \\ 0.9742 \\ 1.0323 \\ 1.0110 \\ 0.9901 \\ 1.0079 \\ 0.9670 \\ 1.0228 \\ 1.0203 \\ 0.9744 \\ 1.0027 \end{bmatrix} \quad \therefore \quad Approximation: \begin{bmatrix} 5 & -3 & 6.6194 \\ 6 & -3 & 7.3196 \\ 7 & -3 & 7.8866 \\ 8 & -3 & 8.8762 \\ 5 & 2 & 6.9553 \\ 6 & 2 & 7.5769 \\ 7 & 2 & 8.1845 \\ 8 & 2 & 9.0410 \\ 5 & 7 & 7.3225 \\ 6 & 7 & 7.8906 \\ 7 & 7 & 8.6000 \\ 8 & 7 & 9.0946 \\ 5 & 12 & 7.6579 \\ 6 & 12 & 8.2930 \\ 7 & 12 & 8.7698 \\ 8 & 12 & 9.5172 \end{bmatrix} \times \begin{bmatrix} C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 0.9785 \\ 0.9938 \\ 0.9726 \\ 1.0672 \\ 0.9904 \\ 0.9842 \\ 0.9742 \\ 1.0323 \\ 1.0110 \\ 0.9901 \\ 1.0079 \\ 0.9670 \\ 1.0228 \\ 1.0203 \\ 0.9744 \\ 1.0027 \end{bmatrix}$$

Equating AX = Pb gives AX = $A(A^TA)^{-1}A^T$ B. Therefore X = $(A^TA)^{-1}A^TB$. So **C** = -0.1764, **D** = -0.0160 and **E** = 0.2738. The model can be completed by the substitution of these values into the original formula (CL + DS +EP = 1) which gives:

$$-0.1764\text{L} - 0.0160\text{S} + 0.2738\text{P} = 1$$

$$so\ P = \frac{1 + 0.1764\boldsymbol{L} + 0.0160\boldsymbol{S}}{0.2738}$$

$$= 3.65 + 0.644L + 0.0584S$$

$$Approximation\ (Model): P = 3.65 + 0.644L + 0.0584S$$

By finding the partial derivatives of P, i.e. once with respect to L (length) and once with respect to S (Strength), we can compare the effect of increasing Length by one to increasing strength by one:

$$\frac{\partial P}{\partial L} = 0.644 \qquad \frac{\partial P}{\partial S} = 0.0584$$

Therefore, an increase of Length by one is much more effective than an increase in strength by one. However, this is expected as I defined strength (how it is calculated), and the steps between each strength are only small increases in strength, i.e. an increase of one strength unit is the same as replacing one card with the card directly above it. In order to get a better, perspective of the effects of length and strength, I will multiply the partial derivative of P with respect to S by 5, which gives me the effect of an increase of one strength category (e.g. from Medium to Strong) on the total score: $0.0584 \times 5 = 0.292$. This leaves an increase in length 2.21 times more effective than an increase in strength category (5 strength units).

In reflection, I will assess the accuracy of this model in two ways. Firstly I will calculate the error that my approximation resulted in, which can be done by finding the differences between P and PB.

B - PB gives the following error vector **E**:

$$\begin{pmatrix} 0.0215 \\ 0.0062 \\ 0.0274 \\ -0.0672 \\ 0.0096 \\ 0.0158 \\ 0.0258 \\ -0.0323 \\ -0.0110 \\ 0.0099 \\ -0.0079 \\ 0.0330 \\ -0.0228 \\ -0.0203 \\ 0.0256 \\ -0.0027 \end{pmatrix}$$

The magnitude of this error is 1.038, which means the error and uncertainty of our model is almost negligible.

Secondly, and finally, in order to apply the model to the real world, i.e. see if it can serve the purpose it was made for, I contacted my expert interviewees once more and presented them with 5 random tarneeb hands, asking them to tell me what they would call for each hand. I also calculated the expected total score using my model. Here are the results:

Tarneeb Length, Tarneeb Strength	Model's calculation of Total Score	Interviewee's Call				
		#1 Izz	#2 Lazy JR	#3 Ahmad	#4 Amjad	**Mode of Interviewees Calls**
5, 12	7.57	7	7	7	7	**7**
5. -7	6.46	Pass	Pass	7	Pass	**Pass**
4, -4	6.00	Pass	Pass	Pass	Pass	**Pass**
6, 16	8.45	8	8	8	10	**8**
8, 13	9.57	9	9	9	`11	**9**

Although more times than not, one of the interviewees gave a different call to the others, the mode of the interviewees' calls was consistently the Model's calculation of the total score rounded down to the nearest full number (nearest call). When the model gave expected scores of less than 7, the mode of the interviewees was a pass, which makes sense as the minimum call allowed in Tarneeb is a 7.

LIMITATIONS OF MODEL

The main limitation of applying our model to the real world is that we assumed that all players are equally good, i.e. expert level, and play with the same set of techniques. In real life, novices are likely to only apply similar techniques to some extent; less experienced players will be less reliable and causes random error. Therefore, this model works best for expert players who will play using techniques stated in the interviews and make no mistakes throughout the game. Not only this, but my model will be limited because we are dealing with human players who have the capacity to think creatively, and even if it is against the rules, there is often interaction

between players. This interaction may be as subtle as noticing another player's reaction to his cards or to finding out which suit is trump and may lead to increased knowledge and a different course of game play.

Also, since I have written the program, I've been learning a lot about tarneeb and playing frequently. I've come up with a couple new strategies, such as "Looting" and increased focus on emptying shortest starting suits (when starting a round) as opposed to playing the "next highest card" of a suit if it is in your possession. These are techniques that other expert players are likely to implement in their games and have not been account for in my model.

CONCLUSION – REFLECTION ON RESULTS

In conclusion, this project has given an exceptional set of results. This can be demonstrated by the strong correlations, with R^2 values often greater than 99% when assessing the effect of our chosen variables, length and strength, on total score. Incorporating the expert strategies into my tarneeb program, I've been able to produce enough data to create a model for making an opening bid: $P = 3.65 + 0.644L + 0.0584S$. Furthermore, the model proved its worth with very small error values and even worked well when tested in the real world. I also had a chance to compare the effect of length and strength on total score by finding partial derivatives of the model; I found that an increase of one in length is over two times more effective on the total score than an increase in one strength category.

BIBLIOGRAPHY:

1. "MEET." MEET. http://meet.mit.edu/ (accessed February 4, 2014).
2. Jabareen, Emad. “Tarneeb”. Google Play Store, Vers. 1.1.0 (2013). https://play.google.com/store/apps/details?id=com.emadoz.tarneeb (accessed February 5, 2014)
3. "Tarneeb." Tarneeb. http://tarneeb.com/ (accessed February 5, 2014).

APPENDIX 1 - INTERVIEWS

IZZ - INTERVIEWEE 1

HOW LONG HAVE YOU BEEN PLAYING TARNEEB

13/17 years

DO YOU PLAY OTHER RELATED CARD GAMES? WHIST, BRIDGE, "AMERICAN", TRIX

Yes, Trix.

DO YOU HAVE ANY PARTICULAR STRATEGIES?

Watch partner's reaction to cards

HOW DO YOU SHUFFLE, CUT?

Random, unless you see a group of good/same suit cards, then cut appropriately

HOW DO YOU DEAL?

13 cards at a time, Increases Competition

HOW DO YOU CHOOSE YOUR CALL? LONG SUITE (X+2)? STRONG SUIT?

Count potential tricks:

6 trumps, 1 high suit= 7/8

5 trumps, 3 other high cards = 7

NO high cards, NO call, UNLESS BLUFF for someone else to call higher

DO YOU COUNT TARNEEB?

Yes, and which numbers are down/left

DO YOU COUNT ANY OTHER SUITS?

Keep track of High cards

ADDITIONAL STRATEGIES?

Intimidate opponents

Figure out other cards of a suit from a play

LAZY JR - INTERVIEWEE 2

HOW LONG HAVE YOU BEEN PLAYING TARNEEB

6/16 years

DO YOU PLAY OTHER RELATED CARD GAMES? WHIST, BRIDGE, "AMERICAN", TRIX

Yes, Fingers.

DO YOU HAVE ANY PARTICULAR STRATEGIES?

Go with the flow...

HOW DO YOU SHUFFLE, CUT?

Cut 1/4, changes cards with opponent (only if deal 13 at a time)

HOW DO YOU DEAL?

13 cards at a time, Increases Competition

HOW DO YOU CHOOSE YOUR CALL? LONG SUITE (X+2)? STRONG SUIT?

Count strongest/longest suite, then holistic judgment assessing high cards of tarneeb and high cards of other suites

Count potential tricks - from own hand

Do not count points from partner/don't rely on partner for any help

DO YOU COUNT TARNEEB?

Yes, and which numbers are down or left in game.

DO YOU COUNT ANY OTHER SUITS?

Keep track of High cards (A,K,Q,J)

ADDITIONAL STRATEGIES?

--

AHMAD - INTERVIEWEE 3

HOW LONG HAVE YOU BEEN PLAYING TARNEEB

46

DO YOU PLAY OTHER RELATED CARD GAMES? WHIST, BRIDGE, "AMERICAN", TRIX

No

DO YOU HAVE ANY PARTICULAR STRATEGIES?

2,3,4,5,A: don't play A

Your call: empty other suits from your hand

3rd Player high

HOW DO YOU SHUFFLE, CUT?

Random

HOW DO YOU DEAL?

NOT 13 at a time. 2 at a time.

HOW DO YOU CHOOSE YOUR CALL? LONG SUITE (X+2)? STRONG SUIT?

long suit < 5: pass

long suit >= 5: call 7+

long suit = 7, call 8+

DO YOU COUNT TARNEEB?

Yes

DO YOU COUNT ANY OTHER SUITS?

High cards

ADDITIONAL STRATEGIES?

--

AMJAD - INTERVIEWEE 4

HOW LONG HAVE YOU BEEN PLAYING TARNEEB

8

DO YOU PLAY OTHER RELATED CARD GAMES? WHIST, BRIDGE, "AMERICAN", TRIX

No

DO YOU HAVE ANY PARTICULAR STRATEGIES?

2,3,4,5,A: don't play A

Never start with Q or J, unless last of suit

If you have a strong long suit that you don't call on, leave to the end to play one after another, loot it (play one after the other)

Always wipe all tarneeb out at the beginning

Doesn't count points

Play ace/king of non-tarneeb

First player play highest card possible if it’s winning

HOW DO YOU SHUFFLE, CUT?

Single "shuffle"/cut. [Cheat]

HOW DO YOU DEAL?

13 at a time

HOW DO YOU CHOOSE YOUR CALL? LONG SUITE (X+2)? STRONG SUIT?

At least 5 for call (7+)

6: call 9

6+1A or 7: call 10

Strong suit over long suit!

2 or more aces... call 7+

DO YOU COUNT TARNEEB?

Yes

DO YOU COUNT ANY OTHER SUITS?

High cards

Know who's out of what suit

ADDITIONAL STRATEGIES?

--

APPENDIX 2 – TARNEEB SIMULATION CODE

```
import random
class autotarneeb:
    #Final
    courts = ("j", "q", "k", 1)
    suits = ("s", "h", "c", "d")
    #User: hand is players[0]
    players = [[],[],[],[]]   # [player0 = [], player1 = [], player2 = [],
player3 = [] ]
    suitQ = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]] #[nofs, nofh, nofc, nofd]
number for each suit.. for each player. suitQ[1][3] is the third suit of player
1.
    #DONE:l_s
    suitQpoints = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]] #Quantity, points
for longest suit
    #DONE :strengths
    suitS =[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]        #strength for each
suit
    #DONE :t_s
    suitD =[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]        #strength
trump_strengtherence
    longest_suit = [None, None, None, None]
    longest_length = [None, None, None, None]
    tarneeb = 0
```

```
    #PLAY
    played_cards = []
    next_highest_card_of_suit = [12,25,38,51]
    currentplay = [None,None,None,None]

    suits_owned_by_player= [[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]]

    rounds_won_by_player = [0,0,0,0]

    def updateDataFields(self, player):
        self.fill_suitQ(player)
        self.strengths(player)
        self.longest_suit[player], self.longest_length[player] =
self.update_longest_suit(player)
        self.diffs(player)
    def strengths(self,player):
        hand = self.players[player]
        spades = self.suitQ[player][0]
        hearts= self.suitQ[player][1]
        clubs= self.suitQ[player][2]
        diamonds= self.suitQ[player][3]
        #Strengths
        strength = 0
        for x in hand[:spades]:
            strength += x%13+2-8
        self.suitS[player][0] = strength
        strength = 0
        for x in hand[spades:spades+hearts]:
            strength += x%13+2-8
        self.suitS[player][1] = strength
        strength = 0
        for x in hand[spades+hearts:spades+hearts+clubs]:
            strength += x%13+2-8
        self.suitS[player][2] = strength
        strength = 0
        for x in hand[spades+hearts+clubs:]:
            strength += x%13+2-8
        self.suitS[player][3] = strength
    def diffs(self,player):
        hand = self.players[player]
        spades,hearts,clubs,diamonds = self.suitQ[player]
        #Strengths calculated for each suit as though that suit were trump
        trump_strength = 0
        for x in hand[:spades]:
            if (x%13)+2<10:
                trump_strength+=1
            else:
                trump_strength += x%13+2-8
        self.suitD[player][0] = trump_strength - self.suitS[player][0]
        trump_strength = 0
        for x in hand[spades:spades+hearts]:
            if (x%13)+2<10:
                trump_strength+=1
            else:
                trump_strength += x%13+2-8
        self.suitD[player][1] = trump_strength - self.suitS[player][1]
        trump_strength = 0
        for x in hand[spades+hearts:spades+hearts+clubs]:
            if (x%13)+2<10:
                trump_strength+=1
            else:
```

```
                trump_strength += x%13+2-8
        self.suitD[player][2] = trump_strength - self.suitS[player][2]
        trump_strength = 0
        for x in hand[spades+hearts+clubs:]:
            if (x%13)+2<10:
                trump_strength+=1
            else:
                trump_strength += x%13+2-8
        self.suitD[player][3] = trump_strength - self.suitS[player][3]

    def update_longest_suit(self, player):
        #UPDATES suitQpoints if updateData == True
        longestLength = max(self.suitQ[player])     #this is the length of the
longest suit
        for q in xrange(4):
            if self.suitQ[player][q] == longestLength:
                self.suitQpoints[player][q] = 1     #assign suitQpoint to suit
which is longest.
                longestSuit = q
        if sum(self.suitQpoints[player]) >1:  # IF there is more than one
"longest suit"
            strengthsoflongest=[]
            for q,p in zip(xrange(4),self.suitQpoints[player]):
                strengthsoflongest.append(p*self.suitS[player][q])
            longestS = max(strengthsoflongest)   #longestS is the larger
strength of the equally long suits.
            for q in xrange(4):
                if self.suitS[player][q]== longestS:
                    longestSuit = q
        else:
            longestSuit = self.suitQpoints[player].index(1)
            longestS = self.suitS[player][longestSuit]
        return (longestSuit, longestLength)
    def fill_suitQ(self,player):
        hand = self.players[player]
        hand.sort()
        spades = 0
        hearts = 0
        clubs = 0
        diamonds = 0
        for card in hand:
            if card < 13:
                spades += 1
            elif card < 26:
                hearts +=1
            elif card < 39:
                clubs += 1
            else:
                diamonds +=1
        self.suitQ[player] = [spades,hearts,clubs,diamonds]

    def current_suit_lengths(self,player):
        current_suitQ = [0,0,0,0]
        for x in self.players[player]:
            if x in range(13):
                current_suitQ[0] += 1
            if x in range(13,26):
                current_suitQ[1] += 1
            if x in range(26,39):
                current_suitQ[2] += 1
            else:
```

```
                current_suitQ[3] += 1
        return current_suitQ
    def current_longest_suit(self,player):
        current_suitQ = self.current_suit_lengths(player)
        return current_suitQ[current_suitQ.index(max(current_suitQ))]  #returns
the suit that is currently longest.

    #Really should only be called for the benefit of the player in the
argument.
    def have_suit(self, player, suit):
        for c in self.players[player]:
            if c >= 13*suit and c < 13*(suit+1):
                return True
        return False
    def other_player_has_suit(self, player, suit):
        return self.suits_owned_by_player[player][suit]

    #checks if the card is already in the "played_cards" list
    def is_card_played(self, card):
        return card in self.played_cards
    def playcardifhave(self,player,card):
        #print "playcardif have", player, card
        have_card_to_play = card in self.players[player]
        if have_card_to_play:
            self.currentplay[player] = card #append ace of tarneeb to
currentplay list
            self.players[player].remove(card) #delete card from players hand
            self.played_cards.append(card)
            if card in self.next_highest_card_of_suit:
                suit_index = self.next_highest_card_of_suit.index(card)
                for i in xrange(1,card%13+1):
                    self.next_highest_card_of_suit[suit_index] = card-i
                    if not self.is_card_played(card-i):
                        break
        return have_card_to_play   #True: played card. #False: Need to look for
alternative

    def check_round_winner(self, round_starter):
        starting_suit = self.currentplay[round_starter]/13
        act_like_no_tarneeb = True  #no trumping going on.
        for played_card in self.currentplay:
            if played_card/13 == self.tarneeb:
                act_like_no_tarneeb = False
        if starting_suit == self.tarneeb:
            act_like_no_tarneeb = True #If starting suit is tarneeb, then no
trumping is going on.
        effective_cards = [-1,-1,-1,-1]
        if not act_like_no_tarneeb:
            starting_suit = self.tarneeb  #tarneeb dominant.
        for card_index in range(4):
            if self.currentplay[card_index]/13 == starting_suit:
                effective_cards[card_index] = self.currentplay[card_index]%13
        return effective_cards.index(max(effective_cards))   #return round
winner.

    def suit_begin_end(self, player, suit):
        suit_begin_index, suit_end_index = None, None
        found_begin_index, found_end_index = False,False
        for index in range(len(self.players[player])):
            if (not found_begin_index) and
self.players[player][index]>=13*suit:  #Entered suit. OR skipped suit.
```

```
                suit_begin_index = index
                found_begin_index = True
            if (not found_end_index) and self.players[player][-index-1] <
13*(suit + 1):  #Entered suit in Reverse.
                suit_end_index = len(self.players[player]) - index
                found_end_index = True
        if suit_begin_index == None:
            return (0,0)
        return (suit_begin_index, suit_end_index)  #returns index of first card
of suit, index of 1+end card of suit.

    #0 means it's the highest card in its suit. #1 means it's the second
highest... etc.
    def rank_card(self, card):
        suit = card/13
        rank = 0
        for x in range(13*(suit+1), 13*suit, -1):
            if card == x:
                return rank
            else:
                if not x in self.played_cards:
                    rank+=1

    #Caller's turn is NOW. He's starting a round
    #Turn order = 0, player = 0
    def caller_first(self):
        #print "first"
        if
self.have_suit(0,self.tarneeb)+self.suits_owned_by_player[1][self.tarneeb] == 2
or self.have_suit(0,self.tarneeb)+self.suits_owned_by_player[1][self.tarneeb] ==
1 and self.suits_owned_by_player[2][self.tarneeb] == 0:  #If it is worth
cleaning tarneeb, from a team perspective
            if self.next_highest_card_of_suit[self.tarneeb] in self.players[0]:
                if
self.playcardifhave(0,self.next_highest_card_of_suit[self.tarneeb]):
                    return
        # if caller not cleaning tarneeb, plays like mortal.
        return self.notcaller_first(0)

    def notcaller_first(self, player):
        #print "nc first"
        for suit in xrange(4):
            if self.suits_owned_by_player[(player+1)%4][suit] and
self.suits_owned_by_player[(player+3)%4][suit] and
self.playcardifhave(player,self.next_highest_card_of_suit[suit]):
                return
        #If you have tarneeb left. Play shortest suit that is not the tarneeb.
Unless your whole hand is tarneeb.
        if self.have_suit(player, self.tarneeb):
            callerSuitQ = self.current_suit_lengths(player)
            callerSuitQ[self.tarneeb] = 99
            for i in xrange(4):
                if callerSuitQ[i] == 0:
                    callerSuitQ[i] = 99
            if min(callerSuitQ) == 99: #only tarneeb left. Play tarneeb.
                if
self.playcardifhave(player,self.players[player][len(self.players[player])-1]):
#PLAY HIGHEST TARNEEB
                    return
            shortestSuit = callerSuitQ.index(min(callerSuitQ))
```

```
            if
self.playcardifhave(player,self.next_highest_card_of_suit[shortestSuit]):
                return
            else:
                for j in range(13*shortestSuit, 13*shortestSuit + 13):
                    if self.playcardifhave(player,j):
                        return
        #If you don't have tarneeb left. THIS WORKS PROPERLY AS OF Feb 22, 2014
        else:
            #If there is a suit your partner can trump, but #4 cannot. Then
play it.
            if self.suits_owned_by_player[(player+2)%4][self.tarneeb]: #(If
partner has tarneeb to begin with)
                for suit in xrange(4):
                    if self.have_suit(player, suit): #can only play a suit you
actually have
                        if self.suits_owned_by_player[(player+3)%4][suit] and
not self.suits_owned_by_player[(player+2)%4][suit]: #fourth has it, Partner
doesn't have it.
                            for j in range(suit*13, 52):   #Don't worry, this
will break before reaching end. Because you DO have the suit.
                                if self.playcardifhave(player, j):  #Play that
card.
                                    return
            #Play a suit everyone has.
            for suit in xrange(4):
                if self.have_suit(player, suit): #Can only play a suit you
actually have.
                    if sum(zip(*self.suits_owned_by_player)[suit]) == 4:#All
players have the suit.
                        for j in range(suit*13, 52): #will break because you DO
have the suit.
                            if self.playcardifhave(player,j):
                                return
            #Otherwise, just play your initially longest suit. It will cost you
little, and cause everyone else to start using up their trumps.
            for k in
xrange(13*self.longest_suit[player],13*self.longest_suit[player]+13):
                if self.playcardifhave(player,k):
                    return
            #If you arrive here, it means you're out of our initial longest
suit. Play current longest.
            suit = self.current_longest_suit(player)
            for j in range(suit*13, 52):
                if self.playcardifhave(player,j):
                    return
        #You REALLY should not get here, but just in case:
        #print "failsafe 1"
        self.playcardifhave(player,self.players[player][0])


    #Turn order = 1
    def second(self, player):
        #print "second", str(self.currentplay)
        suit_to_play = self.currentplay[(player+3)%4]/13   #(player+3)%4 is the
round_starter

        #HAVE THE HIGHEST CARD IN SUIT
        if(self.have_suit(player, suit_to_play)):
```

```
            if
self.playcardifhave(player,self.next_highest_card_of_suit[suit_to_play]):   #pl
ay next highest card of suit
                return

        #TRY: Play small card from suit. Try: play small tarneeb
        trials = [suit_to_play, self.tarneeb]
        for t in trials:
            if self.have_suit(player, t):
                if self.playcardifhave(player,
self.players[player][self.suit_begin_end(player,t)[0]]):
                    return

        #DON'T HAVE THE SUIT. Play small card from one of the remaining two
suits.
        else:
            self.suits_owned_by_player[player][suit_to_play] = 0   #Now everyone
else knows this player is empty of this suit.
            remaining_suits = [s for s in [0,1,2,3] if s != suit_to_play and s!=
self.tarneeb]  #try remaining two suits.
            ranks = []
            for rs in remaining_suits:
                #print rs, self.players[player], self.suit_begin_end(player,
rs)
                ranks.append(self.rank_card(self.players[player][self.suit_begi
n_end(player,rs)[0]]))
            first_card_of_suit_index = self.suit_begin_end(player,
remaining_suits[ranks.index(max(ranks))])[0]
            self.playcardifhave( player, self.players[player][
first_card_of_suit_index ] )

    def third(self, player):
        #print "third",str(self.currentplay)
        suit_to_play = self.currentplay[(player+2)%4]/13
        if self.have_suit(player,suit_to_play): # if you have suit
            beat = False
            suit_begin_index, suit_end_index = self.suit_begin_end(player,
suit_to_play)
            suit_end_index -= 1
            if self.players[player][suit_end_index]-
self.currentplay[(player+2)%4]>1:  #ERROR list index out of range
                for x in
xrange(self.currentplay[(player+2)%4]+2,suit_end_index+1): #for all cards in
between your card and parters   #ERROR List index out of range
                    if x not in self.played_cards:
                        beat = True            if
self.currentplay[0]>self.currentplay[1] and beat: #partner winning, only beat
partner if gap between your highest card
                if self.playcardifhave(player,
self.players[player][13*suit_to_play+self.suitQ[player][suit_to_play]-1]):
                    return
            else: #partner losing or not worth beating
                for v in xrange(13*suit_to_play,13*suit_to_play+13):
                    if self.playcardifhave(player,v):
                        return
        if self.currentplay[1]>self.currentplay[0] and self.have_suit(player,
self.tarneeb):# partner losing -> tarneeb lowest if can
            for v in xrange(13*self.tarneeb,13*self.tarneeb+13):
                if self.playcardifhave(player,v):
                    return
        if not self.suits_owned_by_player[player][self.tarneeb]:
```

```
            for k in
xrange(13*self.longest_suit[player],13*self.longest_suit[player]+max(self.suitQ
[player])):
                if self.playcardifhave(player,k):
                    return
        else: # have tarneeb but partner winning, play lowest card of shortest
suit
            callerSuitQ = self.current_suit_lengths(player)
            callerSuitQ[self.tarneeb] = 99 #Not playing tarneeb just because it
happens to be the shortest suit now.
            for i in xrange(3):
                if callerSuitQ[i] == 0:
                    callerSuitQ[i] = 99
            if min(callerSuitQ) == 99: #only tarneeb left. Play tarneeb.
                if
self.playcardifhave(player,self.players[player][len(self.players[player])-1]):
#PLAY HIGHEST TARNEEB
                    return
            shortestSuit = callerSuitQ.index(min(callerSuitQ))
            for j in range(13*shortestSuit, 13*shortestSuit + 13):
                if self.playcardifhave(player,j):
                    return
        #print "failsafe 3"
        self.playcardifhave(player, self.players[player][0])
    #@returns round_winner
    def fourth(self, player):
        #unlike the other Play functions, this one returns a value.
        #print "fourth",str(self.currentplay)
        suit_to_play = self.currentplay[(player+1)%4]/13   #(player+1)%4 is
round_starter
        if self.have_suit(player,suit_to_play): # if have suit
            if self.currentplay[1]>self.currentplay[0] and
self.currentplay[1]>self.currentplay[2]: #if partner winning
                for v in xrange(13*suit_to_play,13*suit_to_play+13):
                    if self.playcardifhave(player,v):
                        return self.check_round_winner((player+1)%4)
            else: # if partner losing
                for y in xrange(max(self.currentplay), 13*suit_to_play+13): #can
beat in suit
                    if self.playcardifhave(player,y):
                        return self.check_round_winner((player+1)%4)
                for v in xrange(13*suit_to_play,13*suit_to_play+13): # can't
beat-> lowest card of suit
                    if self.playcardifhave(player,v):
                        return self.check_round_winner((player+1)%4)
        else:
            if self.currentplay[1]>self.currentplay[0] and
self.currentplay[1]>self.currentplay[2]: #if partner winning, don't have suit
                if self.suits_owned_by_player[player][self.tarneeb]: # if have
tarneeb, play lowest of shortest suit
                    callerSuitQ = self.current_suit_lengths(player)
                    callerSuitQ[self.tarneeb] = 99 #Not playing tarneeb just
because it happens to be the shortest suit now.
                    for i in xrange(3):
                        if callerSuitQ[i] == 0:
                            callerSuitQ[i] = 99
                    if min(callerSuitQ) == 99: #only tarneeb left. Play tarneeb.
                        if
self.playcardifhave(player,self.players[player][len(self.players[player])-1]):
#PLAY HIGHEST TARNEEB
                            return self.check_round_winner((player+1)%4)
```

```
                    shortestSuit = callerSuitQ.index(min(callerSuitQ))
                    for j in range(13*shortestSuit, 13*shortestSuit + 13):
                        if self.playcardifhave(player,j):
                            return self.check_round_winner((player+1)%4)
                else:
                    for k in
xrange(13*self.longest_suit[player],13*self.longest_suit[player]+13):
                        if self.playcardifhave(player,k):
                            return self.check_round_winner((player+1)%4)
                    current_longest_suit =
self.current_suit_lengths(player).index(max(self.current_suit_lengths(player)))
            else:  #partner losing. Don't have suit. Play tarneeb if have. else
throw away small card.
                if self.have_suit(player, self.tarneeb):  #play smallest tarneeb
that will do the job.
                    tarneeb_range = self.suit_begin_end(player, self.tarneeb)
                    for x in range(*tarneeb_range):
                        if x in self.players[player]:
                            self.currentplay[player] = x
                            if self.check_round_winner((player+1)%4) == player:
                                if self.playcardifhave(player, x):
                                    return player
                #if don't have tarneeb. or don't have large enough tarneeb.
play least valuable card.
                self.suits_owned_by_player[player][suit_to_play] = 0   #Now
everyone else knows this player is empty of this suit.
                remaining_suits = [s for s in [0,1,2,3] if s != suit_to_play and
s!= self.tarneeb]  #try remaining two suits.
                ranks = []
                for rs in remaining_suits:
                    #print rs, self.players[player],
self.suit_begin_end(player, rs)
                    ranks.append(self.rank_card(self.players[player][self.suit_
begin_end(player,rs)[0]]))
                first_card_of_suit_index = self.suit_begin_end(player,
remaining_suits[ranks.index(max(ranks))])[0]
                if self.playcardifhave( player, self.players[player][
first_card_of_suit_index ] ):
                    return self.check_round_winner((player+1)%4)
        #print "failsafe 4"
        if self.playcardifhave(player, self.players[player][0]):
            return self.check_round_winner((player+1)%4)

    def play_game(self, tarneeb):
        self.tarneeb = tarneeb
        self.play_round(0,0)  #so that caller starts game. No rounds played
yet.
    def play_round(self, last_winner, rounds_played):
        self.currentplay = [None,None,None,None]  #print "round", rounds_played,
"last_winner", last_winner
        if rounds_played<13:
            self.play_turn(last_winner,0,rounds_played) #so that last winner
starts round. Turn order starts at 0
        else:
            pass
            #collect point totals, p0, p1...data
    def play_turn(self,round_starter,turn_order, rounds_played):
        player = (round_starter+turn_order)%4
        if turn_order == 0:
            if player == 0:
                self.caller_first()
```

```
                self.play_turn(round_starter, (turn_order+1)%4, rounds_played)
            else:
                self.notcaller_first(player)
                self.play_turn(round_starter, (turn_order+1)%4, rounds_played)
        elif turn_order == 1:
            self.second(player)
            self.play_turn(round_starter, (turn_order+1)%4, rounds_played)
        elif turn_order == 2:
            self.third(player)
            self.play_turn(round_starter, (turn_order+1)%4, rounds_played)
        else:
            winner_of_this_round = self.fourth(player)
            self.rounds_won_by_player[winner_of_this_round] += 1
            self.play_round(winner_of_this_round,rounds_played+1)  #start new
round


    def play_four_games(self):
        deck = range(52)
        random.shuffle(deck)
        for x in xrange(4):
            for i in xrange(4):
                self.players[i] = deck[i*13:13*(i+1)]
                self.players[i].sort()
                self.updateDataFields(i)
                self.rounds_won_by_player = [0,0,0,0]
            self.fo.write("hand"+ str(self.players[0]) +"\n")
            self.fo.write("strengths"+ str(self.suitS[0]) +"\n")
            self.fo.write("lengths"+ str(self.suitQ[0]) +"\n")
            self.fo.write("diffs"+ str(self.suitD[0]) +"\n")
            self.fo.write("longest suit"+ str(self.longest_suit[0]) +"\n")
            self.fo.write("longest suit length"+ str(self.longest_length) +"\n")
            self.fo.write("tarneeb"+ str(x) +"\n")
            self.fo.write("tarneeb length"+str(self.suitQ[0][x]) + "\n")
            self.fo.write("tarneeb strength"+str(self.suitS[0][x]) + "\n")
            self.fo.write("tarneeb diff" + str(self.suitD[0][x]) + "\n")
            self.play_game(x)
            self.fo.write("caller won" + str(self.rounds_won_by_player[0]) +
"\n")
            self.fo.write("partner won" + str(self.rounds_won_by_player[2]) +
"\n")
            self.fo.write("total score" + str(self.rounds_won_by_player[0] +
self.rounds_won_by_player[2]) + "\n")

    def __init__ (self):
        for x in range(100):
            print x
            self.fo = open("data5.txt", "a")
            for i in range(100):
                self.play_four_games()
            self.fo.close()


autotarneeb()
```