Basic QPQ vs Multilevel QPQ

Import some libraries

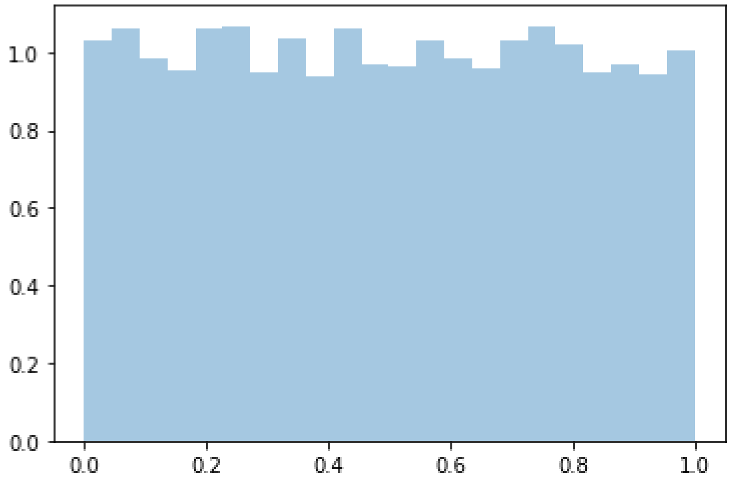
In [1]:
```
%matplotlib inline

import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import beta
from scipy.ndimage.interpolation import shift
import seaborn as sns
```

The following example shows how the liar's values are calculated. The idea is that declared values are strongly correlated with real values but slightly increase the probability of saying higher values. The way to do it is by combining the CDF of the uniform and beta distributions.

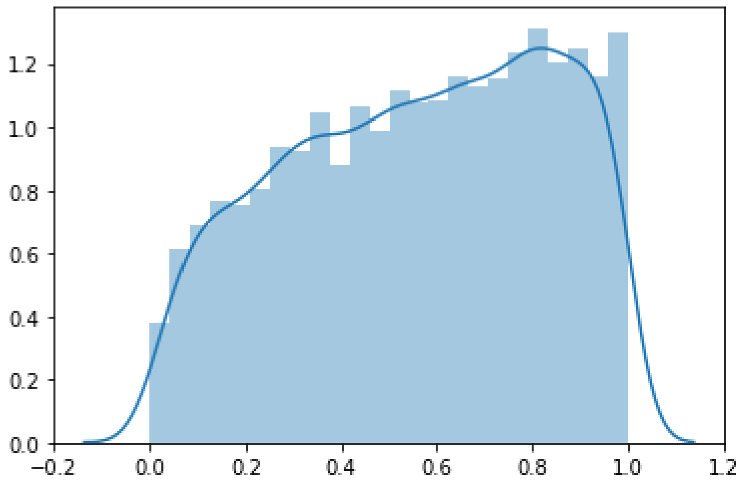
The following code generates true values following a uniform distribution.

In [2]:
```
trueVals = stats.uniform(0, 1).rvs(10000)
sns.distplot(trueVals, kde=False, norm_hist=True);
```



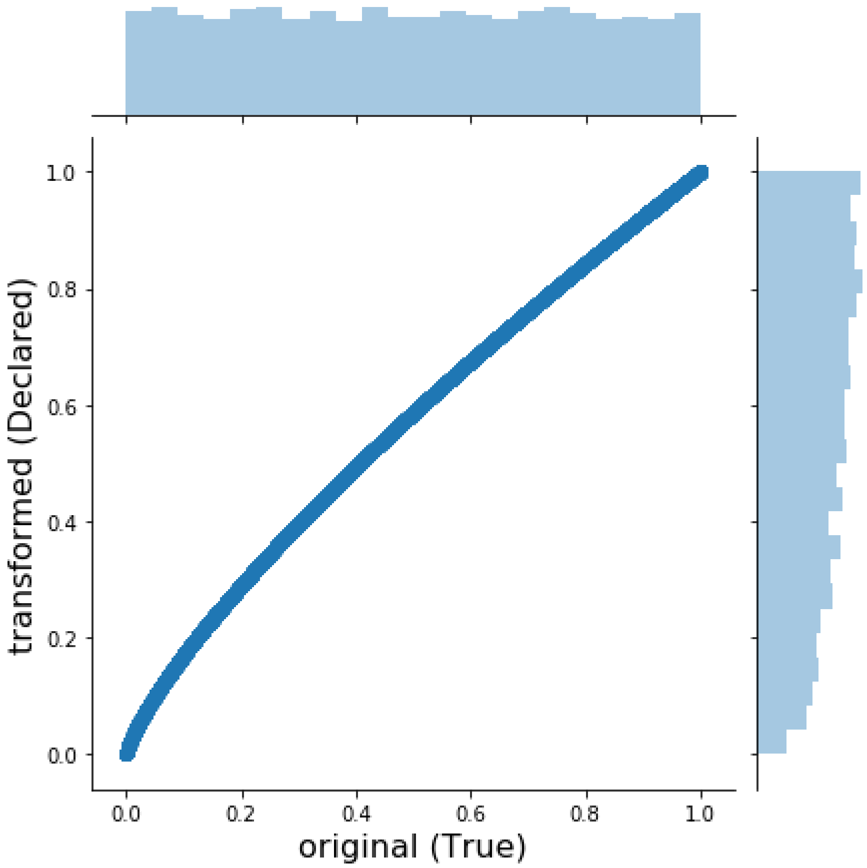
Now, transforms the uniform values (true values) using the beta distribution.

```
In [3]: norm = stats.distributions.beta(1.3, 1)
        declaredVals = norm.ppf(trueVals)
        sns.distplot(declaredVals);
```



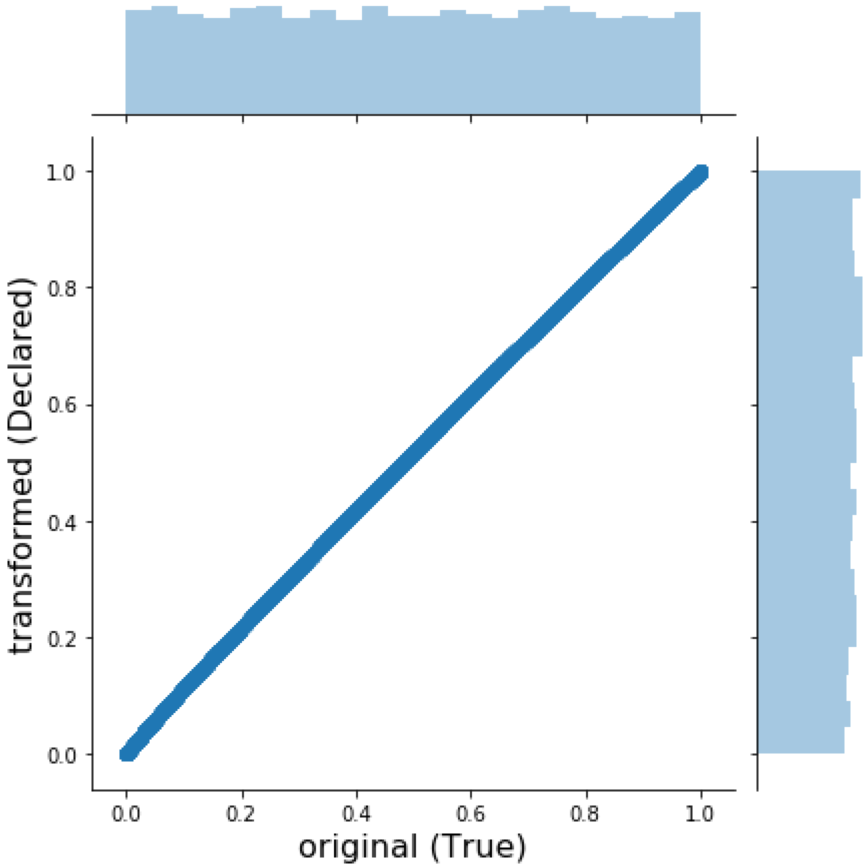
It is easy to see that declared values and true values are highly correlated, but higher values are slightly more frequent. Given that QPQ chooses low values for decisions (argmin), one possible strategy for liars would be to modify the actual values by declaring higher values more likely.

```
In [4]: h = sns.jointplot(trueVals, declaredVals, stat_func=None)
        h.set_axis_labels('original (True)', 'transformed (Declared)', fontsize=16);
```



The previous example exaggerates the beta value in order to show the technique, but the real strategy is to use lower Beta values to make liars detection more complicated. The following example could be a case of almost undetectable manipulation

In [5]:
```
norm = stats.distributions.beta(1.05, 1)
declaredVals = norm.ppf(trueVals)
h = sns.jointplot(trueVals, declaredVals, stat_func=None)
h.set_axis_labels('original (True)', 'transformed (Declared)', fontsize=16);
```



About KS Test

[More info at scipy.stats Web page (https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html)](https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.kstest.html)

scipy.stats.kstest(rvs, cdf, args=(), N=20, alternative='two-sided', mode='approx')

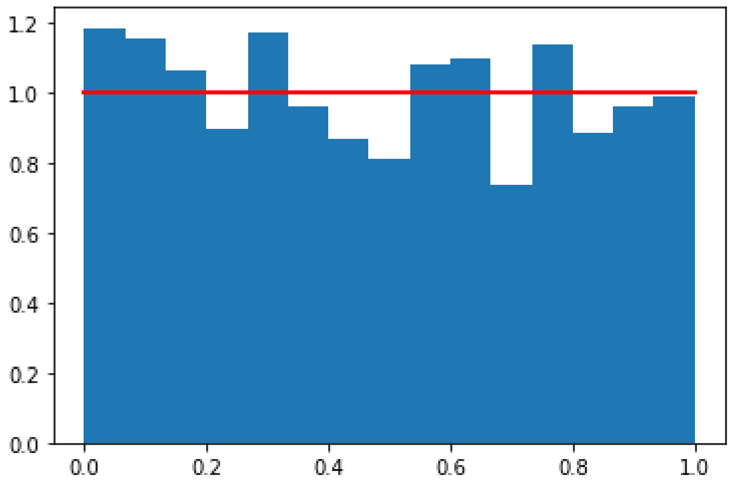
Perform the Kolmogorov-Smirnov test for goodness of fit. This performs a test of the distribution G(x) of an observed random variable against a given distribution F(x). Under the null hypothesis the two distributions are identical, G(x)=F(x). The alternative hypothesis can be either ‘two-sided’ (default), ‘less’ or ‘greater’. The KS test is only valid for continuous distributions.

Draw samples from the uniform distribution:

In [6]:
```
s = np.random.uniform(0, 1, 1000)
```

Display the histogram of the samples, along with the probability density function:

In [7]:
```
count, bins, ignored = plt.hist(s, 15, density=True)
plt.plot(bins, np.ones_like(bins), linewidth=2, color='r')
plt.show()
```



Test against ‘two-sided’ alternative hypothesis uniform. The null-hypothesis for the KT test is that the distributions are the same. Thus, the lower your p value the greater the statistical evidence you have to reject the null hypothesis and conclude the distributions are different.

In [8]:
```
stats.kstest(s, 'uniform')
```

Out[8]:
```
KstestResult(statistic=0.03348780075573765, pvalue=0.2076068016214482)
```

Computing False-Negatives proportion

In [9]:

```
# Number of simulations
numberSimulation = 50000

# History length
historyLen = 100

pvalues = []
for i in range(numberSimulation):
    # s = np.random.beta(1, 1.1, historyLen)
    s = np.random.uniform(0, 1, historyLen)
    ks = stats.kstest(s, 'uniform')
    pvalues.append(ks)

df = pd.DataFrame(data=pvalues)
df.head(10)
```

Out[9]:

	statistic	pvalue
0	0.080409	0.520687
1	0.069130	0.735184
2	0.050905	0.957861
3	0.071680	0.683082
4	0.069356	0.730492
5	0.071821	0.680255
6	0.108431	0.176985
7	0.090256	0.369479
8	0.101071	0.242305
9	0.151515	0.018075

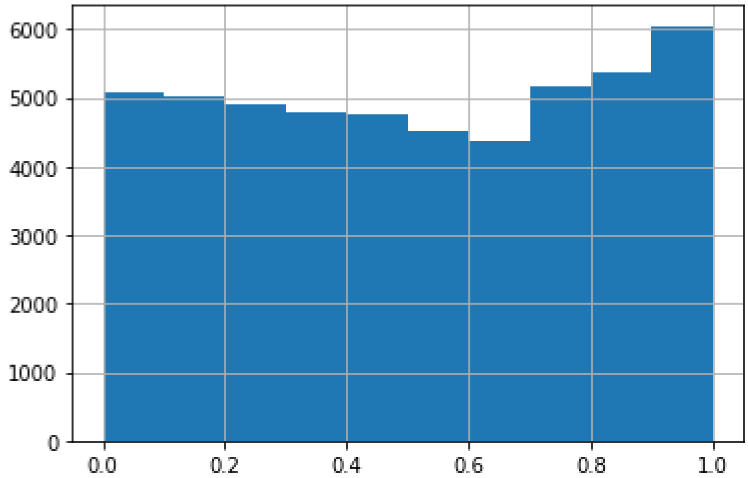
In [10]:

```
df2 = df[df.pvalue < 0.2]['pvalue']
len(df2)
```

Out[10]: 10079

In [11]:

```
hist = df['pvalue'].hist()
```



Terminology

First, let's introduce some terminology

- **condition positive (P)**: the number of real positive cases in the data (number of true values. Players are honest)
- **condition negative (N)**: the number of real negative cases in the data (number of false declarations. Players are dishonest)
- **true positive (TP)**: These are cases in which we predicted yes (they are honest), and they are honest. Also known as a hit
- **true negative (TN)**: We predicted no (they are dishonest), and they are dishonest. Also known as a correct anwser.
- **false positive (FP)**: We predicted yes (they are honest), but they are dishonest. Also known as a "Type I error" or false alarm.
- **false negative (FN)**: We predicted no (they are dishonest), but they actually are honest. Also known as a miss or "Type II error"

We are interested on:

- reducing FN rate. Each FN reduces the utility of the player (if we use argmax in QPQ or increases when using argmin).
- increasing TN rate. Dishonest player should be detected as much as possible.

KS test at Level 1

This analysis is within a cluster (inferior level or level 1).

Honest Players

At this point, we want to compute TP and FN when every player is honest. In this case, N, TN and FP are cero. And we want to compute the TP and FN rates at different history lengths.

The goal is to have FN as low as possible.

In [12]:

```python
# Number of simulations
numberSimulation = 5000

# History length
historyLenRange = range(100, 2100, 100)

thresholdRange = range(75, 100, 5)

debug = False

results = {'HL': [],'TH': [], 'P': [],'N': [],
           'TP': [], 'TN': [], 'FP': [], 'FN': [],
           'TPRate': [], 'TNRate': [], 'FPRate': [], 'FNRate': []}
for threshold in thresholdRange:
    threshold = threshold / 100
    print("Threshold:", threshold)
    for historyLen in historyLenRange:
        #print("   historyLen:", historyLen)
        pvalues = []
        for i in range(numberSimulation):
            # Draw samples from a uniform distribution.
            # Samples are uniformly distributed over the half-open interval [low, hig
h) (includes low, but excludes high).
            s = np.random.uniform(low=0, high=1, size=historyLen)
            ks = stats.kstest(s, 'uniform')
            pvalues.append(ks)

        df = pd.DataFrame(data=pvalues)
        TP = len(df[df.pvalue < threshold])
        FN = len(df[df.pvalue >= threshold])
        if debug:
            print("      False negative:", (FN))
            print("      True positive:", (TP))
            assert (numberSimulation == FN + TP), "The number of simulations does not
match!"
            print("      Threshold:", threshold)
            print("      False negative rate:", (FN/numberSimulation))
            print("      True positive rate:", (TP/numberSimulation))
        results['HL'].append(historyLen)
        results['TH'].append(threshold)
        results['P'].append(numberSimulation)
        results['N'].append(0)
        results['TP'].append(TP)
        results['TN'].append(0)
        results['FP'].append(0)
        results['FN'].append(FN)
        results['TPRate'].append(TP/numberSimulation)
        results['TNRate'].append(0)
        results['FPRate'].append(0)
        results['FNRate'].append(FN/numberSimulation)
    df = pd.DataFrame(data=results)
```

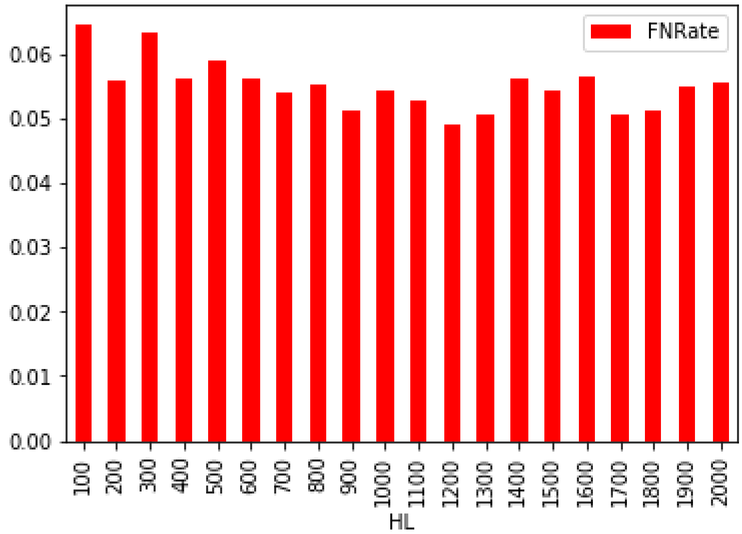
```
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
```

In [13]:
```
df.head(10)
```

Out[13]:

	HL	TH	P	N	TP	TN	FP	FN	TPRate	TNRate	FPRate	FNRate
0	100	0.75	5000	0	3483	0	0	1517	0.6966	0	0	0.3034
1	200	0.75	5000	0	3623	0	0	1377	0.7246	0	0	0.2754
2	300	0.75	5000	0	3665	0	0	1335	0.7330	0	0	0.2670
3	400	0.75	5000	0	3685	0	0	1315	0.7370	0	0	0.2630
4	500	0.75	5000	0	3682	0	0	1318	0.7364	0	0	0.2636
5	600	0.75	5000	0	3703	0	0	1297	0.7406	0	0	0.2594
6	700	0.75	5000	0	3690	0	0	1310	0.7380	0	0	0.2620
7	800	0.75	5000	0	3746	0	0	1254	0.7492	0	0	0.2508
8	900	0.75	5000	0	3711	0	0	1289	0.7422	0	0	0.2578
9	1000	0.75	5000	0	3687	0	0	1313	0.7374	0	0	0.2626

In [14]:
```
df95 = df[df.TH == 0.95]
df95.plot(kind='bar',x='HL',y='FNRate',color='red')
plt.show()
```

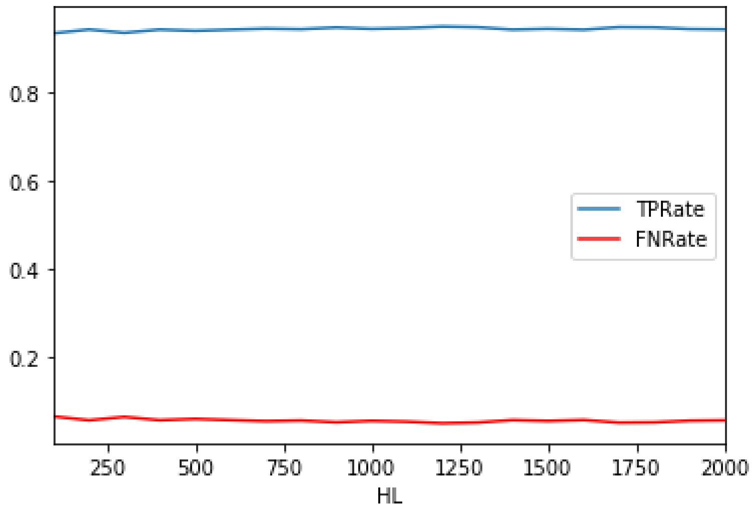


As we can see, the FN rate is constant regardless of the history length.

In [15]:
```python
ax = plt.gca()

df95.plot(kind='line',x='HL',y='TPRate',ax=ax)
df95.plot(kind='line',x='HL',y='FNRate', color='red', ax=ax)

plt.show()
```



And it is also easy to see that:

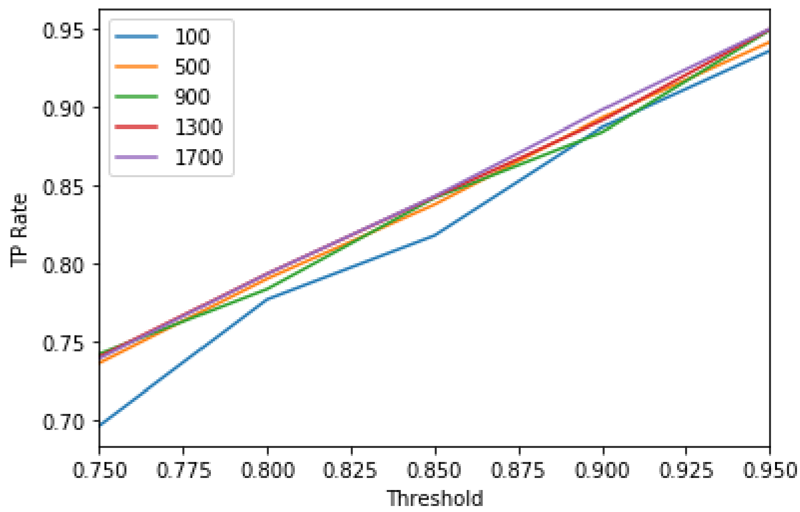
$$TP = threshold$$
$$FN = 1 - threshold$$

In [16]:
```python
ax = plt.gca()

for HL in  range(100, 2100, 400):
    dfTmp = df[df.HL == HL]
    dfTmp.plot(kind='line',x='TH',y='TPRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("TP Rate")
ax.legend(loc='best')

plt.show()
```

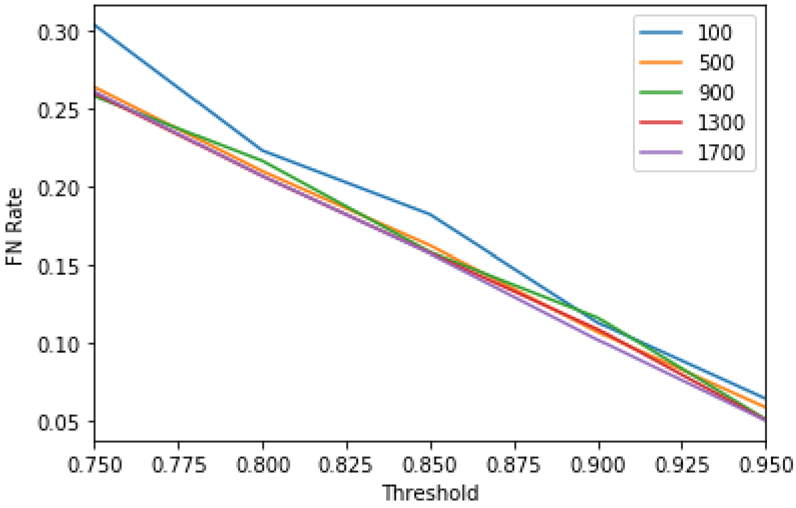


In [17]:
```
ax = plt.gca()

for HL in  range(100, 2100, 400):
    dfTmp = df[df.HL == HL]
    dfTmp.plot(kind='line',x='TH',y='FNRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("FN Rate")
ax.legend(loc='best')

plt.show()
```



Dishonest Players

Now, we want to compute the confusion matrix when the player is dishonet following a beta. In this case, P, TP and FN are cero.

And we want to compute those TN and FP rates with different history lengths.

The goal is to have TP as high as possible.

In [24]:

```python
# Number of simulations
numberSimulation = 5000

# History length
historyLenRange = range(10, 100, 10) #  range(100, 2100, 400)

thresholdRange = range(75, 100, 5)

betaFactor = 1.05
betaDistr = stats.distributions.beta(betaFactor, 1)

debug = False

results = {'HL': [],'TH': [], 'P': [],'N': [],
           'TP': [], 'TN': [], 'FP': [], 'FN': [],
           'TPRate': [], 'TNRate': [], 'FPRate': [], 'FNRate': []}
for threshold in thresholdRange:
    threshold = threshold / 100
    print("Threshold:", threshold)
    for historyLen in historyLenRange:
        #print("   historyLen:", historyLen)
        pvalues = []
        for i in range(numberSimulation):
            trueVals = np.random.uniform(0, 1, historyLen)
            declaredVals = betaDistr.ppf(trueVals)
            ks = stats.kstest(declaredVals, 'uniform')
            pvalues.append(ks)

        df = pd.DataFrame(data=pvalues)
        FP = len(df[df.pvalue < threshold])
        TN = len(df[df.pvalue >= threshold])
        if debug:
            print("      False positive:", (FP))
            print("      True negative:", (TN))
            assert (numberSimulation == FP + TN), "The number of simulations does not
match!"
            print("      Threshold:", threshold)
            print("      False positive rate:", (FP/numberSimulation))
            print("      True negative rate:", (TN/numberSimulation))
        results['HL'].append(historyLen)
        results['TH'].append(threshold)
        results['P'].append(0)
        results['N'].append(numberSimulation)
        results['TP'].append(0)
        results['TN'].append(TN)
        results['FP'].append(FP)
        results['FN'].append(0)
        results['TPRate'].append(0)
        results['TNRate'].append(TN/numberSimulation)
        results['FPRate'].append(FP/numberSimulation)
        results['FNRate'].append(0)
    df = pd.DataFrame(data=results)
```

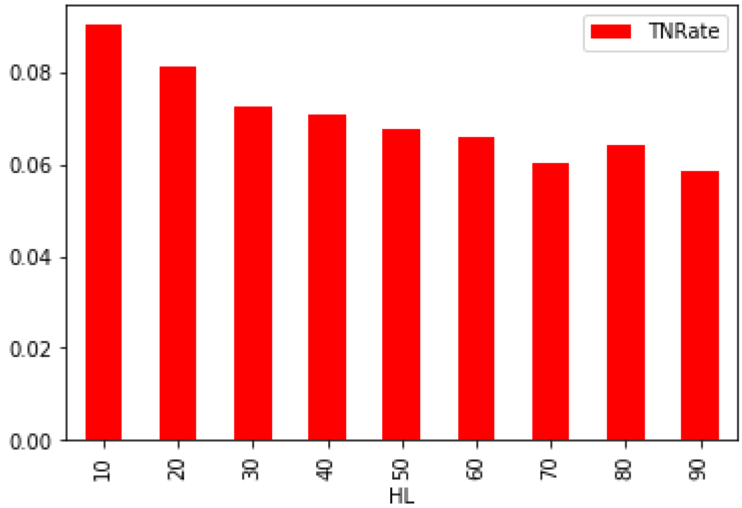
```
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
```

In [25]:
```
df.head(10)
```

Out[25]:

	HL	TH	P	N	TP	TN	FP	FN	TPRate	TNRate	FPRate	FNRate
0	10	0.75	0	5000	0	1405	3595	0	0	0.2810	0.7190	0
1	20	0.75	0	5000	0	1386	3614	0	0	0.2772	0.7228	0
2	30	0.75	0	5000	0	1394	3606	0	0	0.2788	0.7212	0
3	40	0.75	0	5000	0	1389	3611	0	0	0.2778	0.7222	0
4	50	0.75	0	5000	0	1375	3625	0	0	0.2750	0.7250	0
5	60	0.75	0	5000	0	1369	3631	0	0	0.2738	0.7262	0
6	70	0.75	0	5000	0	1350	3650	0	0	0.2700	0.7300	0
7	80	0.75	0	5000	0	1319	3681	0	0	0.2638	0.7362	0
8	90	0.75	0	5000	0	1369	3631	0	0	0.2738	0.7262	0
9	10	0.80	0	5000	0	1345	3655	0	0	0.2690	0.7310	0

In [26]:
```
df95 = df[df.TH == 0.95]
df95.plot(kind='bar',x='HL',y='TNRate',color='red')
plt.show()
```



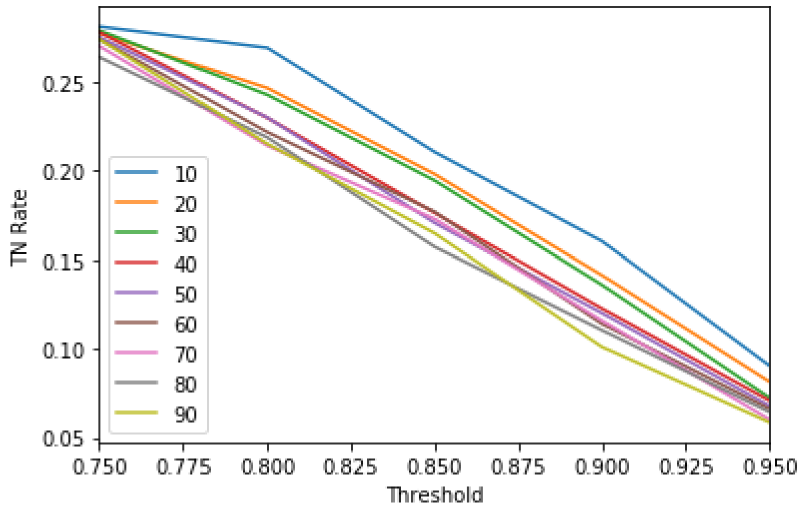
In [28]:

```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[df.HL == HL]
    dfTmp.plot(kind='line',x='TH',y='TNRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("TN Rate")
ax.legend(loc='best')

plt.show()
```



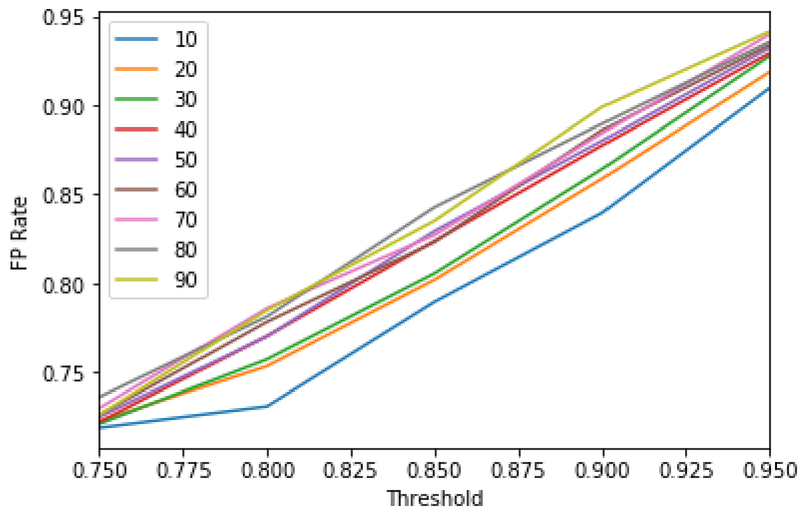
In [29]:

```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[df.HL == HL]
    dfTmp.plot(kind='line',x='TH',y='FPRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("FP Rate")
ax.legend(loc='best')

plt.show()
```



KS test at Level 0

Now, we do the same analysis but at level 0 (superior level where players are clusters)

Honest Clusters

At this point, we want to compute TP and FN when every cluster is honest. In this case, N, TN and FP are cero. And we want to compute the TP and FN rates at different history lengths.

The goal is to have FN as low as possible.

In [30]:
```
# Number of simulations
numberSimulation = 5000

# History length
historyLenRange = range(100, 2100, 100)

thresholdRange = range(75, 100, 5)

debug = False

playersPerClusterRange = range(4, 20, 4)

results = {'players': [], 'HL': [], 'TH': [], 'P': [],'N': [],
           'TP': [], 'TN': [], 'FP': [], 'FN': [],
           'TPRate': [], 'TNRate': [], 'FPRate': [], 'FNRate': []}
for playersPerCluster in playersPerClusterRange:
    print("Players Per Cluster:", playersPerCluster)
    for threshold in thresholdRange:
        threshold = threshold / 100
        print("  Threshold:", threshold)
        for historyLen in historyLenRange:
            #print("   historyLen:", historyLen)
            pvalues = []
            for i in range(numberSimulation):
                s =  np.random.beta(a=1., b=playersPerCluster, size=historyLen)
                ks = stats.kstest(s, stats.beta(a=1., b=playersPerCluster).cdf)
                pvalues.append(ks)

            df = pd.DataFrame(data=pvalues)
            TP = len(df[df.pvalue < threshold])
            FN = len(df[df.pvalue >= threshold])
            if debug:
                print("      False negative:", (FN))
                print("      True positive:", (TP))
                assert (numberSimulation == FN + TP), "The number of simulations does
not match!"
                print("      Threshold:", threshold)
                print("      False negative rate:", (FN/numberSimulation))
                print("      True positive rate:", (TP/numberSimulation))
            results['players'].append(playersPerCluster)
            results['HL'].append(historyLen)
            results['TH'].append(threshold)
            results['P'].append(numberSimulation)
            results['N'].append(0)
            results['TP'].append(TP)
            results['TN'].append(0)
            results['FP'].append(0)
            results['FN'].append(FN)
            results['TPRate'].append(TP/numberSimulation)
            results['TNRate'].append(0)
            results['FPRate'].append(0)
            results['FNRate'].append(FN/numberSimulation)
        df = pd.DataFrame(data=results)
```

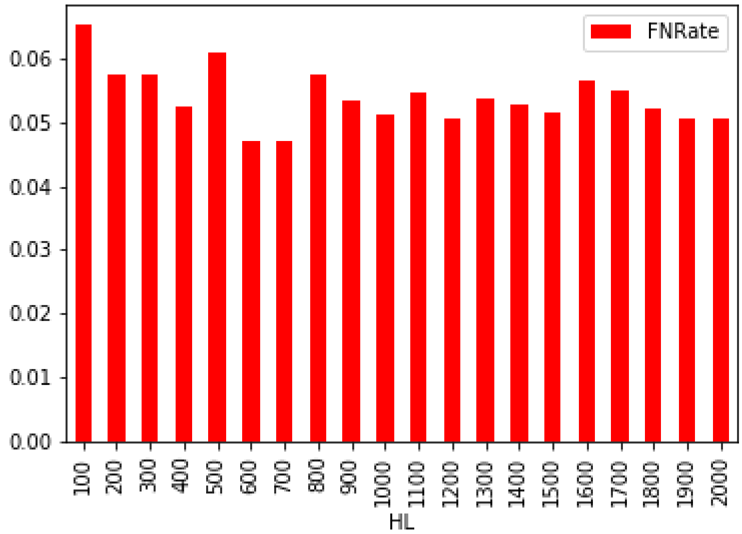
```
Players Per Cluster: 4
  Threshold: 0.75
  Threshold: 0.8
  Threshold: 0.85
  Threshold: 0.9
  Threshold: 0.95
Players Per Cluster: 8
  Threshold: 0.75
  Threshold: 0.8
  Threshold: 0.85
  Threshold: 0.9
  Threshold: 0.95
Players Per Cluster: 12
  Threshold: 0.75
  Threshold: 0.8
  Threshold: 0.85
  Threshold: 0.9
  Threshold: 0.95
Players Per Cluster: 16
  Threshold: 0.75
  Threshold: 0.8
  Threshold: 0.85
  Threshold: 0.9
  Threshold: 0.95
```

In [31]:
```
df.head(10)
```

Out[31]:

	players	HL	TH	P	N	TP	TN	FP	FN	TPRate	TNRate	FPRate	FNRate
0	4	100	0.75	5000	0	3579	0	0	1421	0.7158	0	0	0.2842
1	4	200	0.75	5000	0	3602	0	0	1398	0.7204	0	0	0.2796
2	4	300	0.75	5000	0	3691	0	0	1309	0.7382	0	0	0.2618
3	4	400	0.75	5000	0	3711	0	0	1289	0.7422	0	0	0.2578
4	4	500	0.75	5000	0	3686	0	0	1314	0.7372	0	0	0.2628
5	4	600	0.75	5000	0	3684	0	0	1316	0.7368	0	0	0.2632
6	4	700	0.75	5000	0	3709	0	0	1291	0.7418	0	0	0.2582
7	4	800	0.75	5000	0	3662	0	0	1338	0.7324	0	0	0.2676
8	4	900	0.75	5000	0	3691	0	0	1309	0.7382	0	0	0.2618
9	4	1000	0.75	5000	0	3685	0	0	1315	0.7370	0	0	0.2630

In [32]:
```
dfp4th95 = df[(df.TH == 0.95) & (df.players == 16)]
dfp4th95.plot(kind='bar',x='HL',y='FNRate',color='red')
plt.show()
```

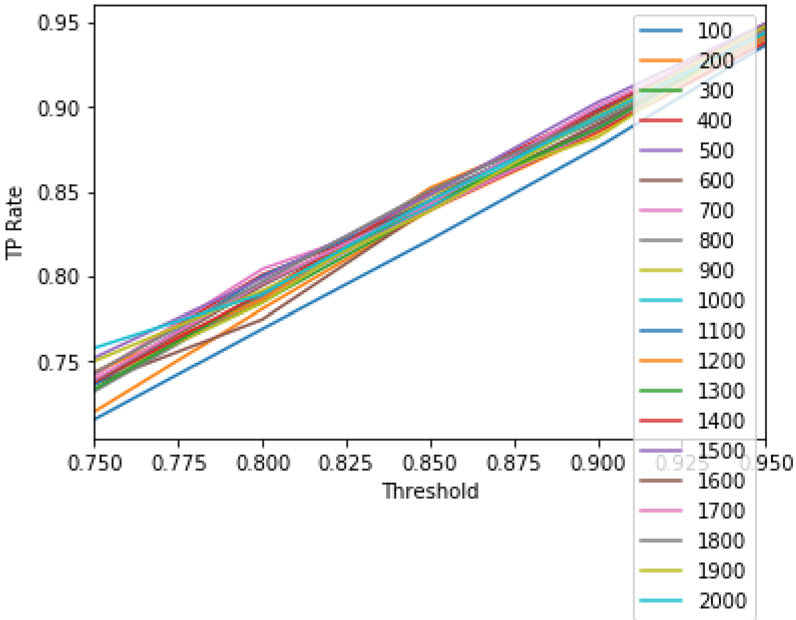


In [33]:
```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[(df.HL == HL) & (df.players == 4)]
    dfTmp.plot(kind='line',x='TH',y='TPRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("TP Rate")
ax.legend(loc='best')

plt.show()
```

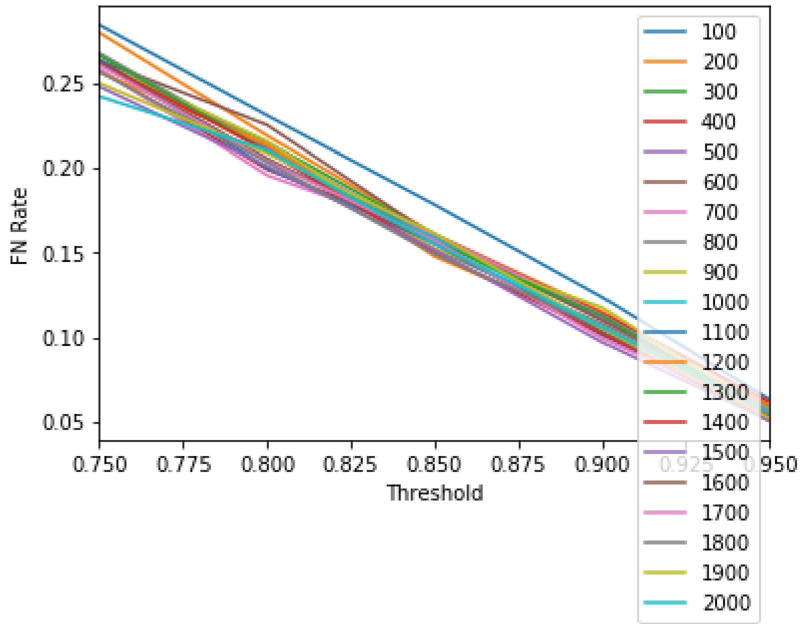


In [34]:
```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[(df.HL == HL) & (df.players == 4)]
    dfTmp.plot(kind='line',x='TH',y='FNRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("FN Rate")
ax.legend(loc='best')

plt.show()
```



Dishonest Clusters

Now, we want to compute the confusion matrix when the cluster is dishonet following a beta with higher alpha parameter. In this case, P, TP and FN are cero.

And we want to compute those TN and FP rates with different history lengths.

The goal is to have TP as high as possible.

In [35]:
```
# Number of simulations
numberSimulation = 5000

# History length
historyLenRange = range(100, 2100, 100)

thresholdRange = range(75, 100, 5)

betaFactor = 1.1

debug = False

playersPerClusterRange = range(4, 20, 4)

results = {'players': [], 'HL': [], 'TH': [], 'P': [],'N': [],
           'TP': [], 'TN': [], 'FP': [], 'FN': [],
           'TPRate': [], 'TNRate': [], 'FPRate': [], 'FNRate': []}
for playersPerCluster in playersPerClusterRange:
    print("Players Per Cluster:", playersPerCluster)
    for threshold in thresholdRange:
        threshold = threshold / 100
        print("Threshold:", threshold)
        for historyLen in historyLenRange:
            #print("   historyLen:", historyLen)
            pvalues = []
            for i in range(numberSimulation):
                trueVals =  np.random.beta(a=1., b=playersPerCluster, size=historyLen
)
                betaDistr = stats.distributions.beta(betaFactor, playersPerCluster)
                declaredVals = betaDistr.ppf(trueVals)
                ks = stats.kstest(s, stats.beta(a=1., b=playersPerCluster).cdf)
                pvalues.append(ks)

            df = pd.DataFrame(data=pvalues)
            FP = len(df[df.pvalue < threshold])
            TN = len(df[df.pvalue >= threshold])
            if debug:
                print("      False positive:", (FP))
                print("      True negative:", (TN))
                assert (numberSimulation == FP + TN), "The number of simulations does
not match!"
                print("      Threshold:", threshold)
                print("      False positive rate:", (FP/numberSimulation))
                print("      True negative rate:", (TN/numberSimulation))
            results['players'].append(playersPerCluster)
            results['HL'].append(historyLen)
            results['TH'].append(threshold)
            results['P'].append(0)
            results['N'].append(numberSimulation)
            results['TP'].append(0)
            results['TN'].append(TN)
            results['FP'].append(FP)
            results['FN'].append(0)
            results['TPRate'].append(0)
            results['TNRate'].append(TN/numberSimulation)
            results['FPRate'].append(FP/numberSimulation)
            results['FNRate'].append(0)
        df = pd.DataFrame(data=results)
```

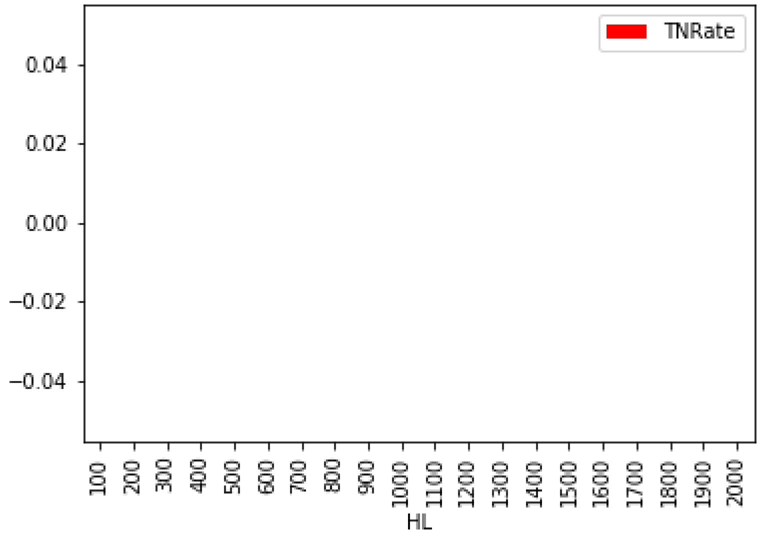
```
Players Per Cluster: 4
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
Players Per Cluster: 8
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
Players Per Cluster: 12
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
Players Per Cluster: 16
Threshold: 0.75
Threshold: 0.8
Threshold: 0.85
Threshold: 0.9
Threshold: 0.95
```

In [36]:
```
df.head(10)
```

Out[36]:

	players	HL	TH	P	N	TP	TN	FP	FN	TPRate	TNRate	FPRate	FNRate
0	4	100	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
1	4	200	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
2	4	300	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
3	4	400	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
4	4	500	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
5	4	600	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
6	4	700	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
7	4	800	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
8	4	900	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0
9	4	1000	0.75	0	5000	0	0	5000	0	0	0.0	1.0	0

In [37]:
```
df95 = df[(df.TH == 0.95) & (df.players == 8)]
df95.plot(kind='bar',x='HL',y='TNRate',color='red')
plt.show()
```

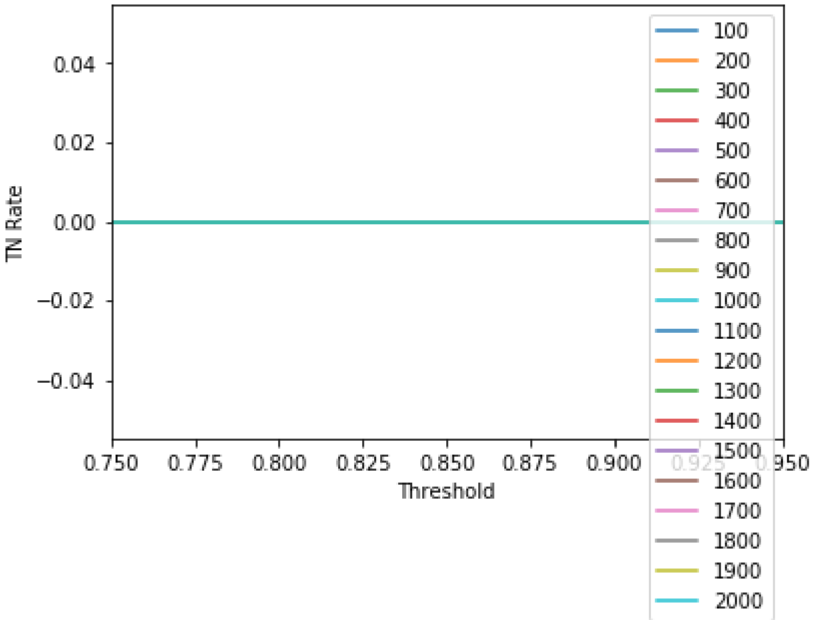


In [38]:
```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[(df.HL == HL) & (df.players == 8)]
    dfTmp.plot(kind='line',x='TH',y='TNRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("TN Rate")
ax.legend(loc='best')

plt.show()
```

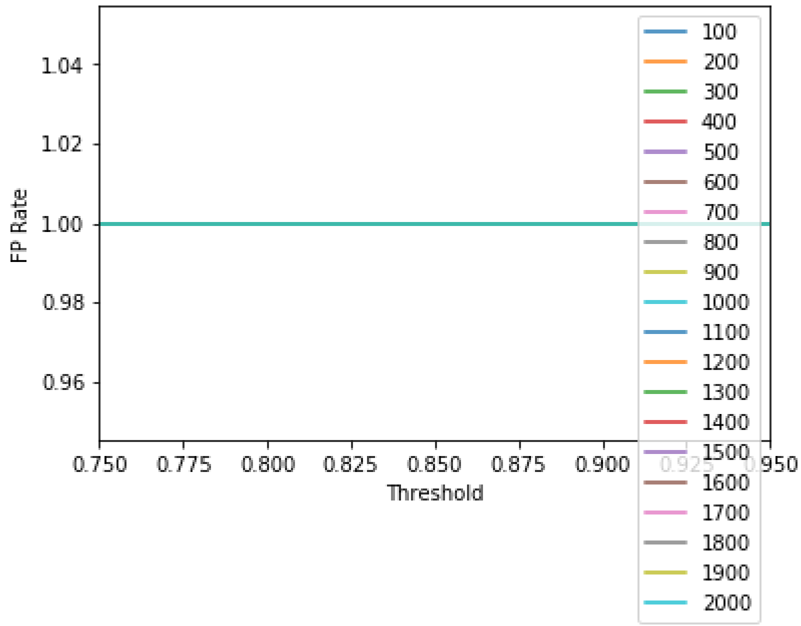


In [39]:
```
ax = plt.gca()

for HL in historyLenRange:
    dfTmp = df[(df.HL == HL) & (df.players == 4)]
    dfTmp.plot(kind='line',x='TH',y='FPRate',ax=ax,label=HL)


ax.set_xlabel("Threshold")
ax.set_ylabel("FP Rate")
ax.legend(loc='best')

plt.show()
```



Threshold Function

So, we know that the FN rate functions for QPQ and QPQ2 are:

FN Rate of QPQ = 1 - th <- independent of historyLen (for the p-value of the KS test) FN Rate of QPQ2 = 1 - thc * thu <- independent of historyLen (for the p-value of the KS test)

Making the two thresholds at QPQ2 $thc = thu = th2$

$FN_{QPQ1} = 1 - th$

$FN_{QPQ2} = 1 - (th2) * (th2)$

$FN_{QPQ1} = FN_{QPQ2}$

$1 - th = 1 - (th2) * (th2)$

$th = th2 * *2$

$th2 = sqrt(th)$

We call $f(x)$ the treshold function that given a history len returns the threshold. This function also gives the probability of not having an FN if the story has length x.

We want that using the same amount of memory, when all are honest, the probability of an FN is the same in QPQ and QPQ2:

$f((historyLen * numplayers)/(numclusters + alpha * numplayers/numclusters)) = sqrt(f(historyLen$

Because in each player with QPQ numplayers historyLen memory is used. In QPQ2 I have

$(historyLen * numplayers)/(numclusters + alpha * numplayers/numclusters)$

if

$x = historyLen$

$a = numplayers * numclusters/(numclusters^2 + numplayers)$

then

$f(ax) = sqrt(f(x))$

Resolving ...

$f(x) = e^{c2^{1-log(x)/log(a)}}$

That when $c < 0$ has the form we want (increasing with x asymptotically 1)

https://www.wolframalpha.com/input/?i=plot+%7Be+%5E+%7B-2%5E%281+-+log%28x%29%2Flog%285%29%29%7D+%7D+from+10+to+5000 (https://www.wolframalpha.com/input/?i=plot+%7Be+%5E+%7B-2%5E%281+-+log%28x%29%2Flog%285%29%29%7D+%7D+from+10+to+5000)

To choose c we can set the threshold value to th_0 for a fixed history length value $historyLen = x_0$

The value of c depends on a, th_0 and x_0 and has the expression:

$c = ln(th_0) * 2^{-1+log(x_0)/log(a)}$

That is always negative as we want when $th_0 < 1$

Entering c in the previous expression:

$$f(x) = e^{ln(th_0) * 2^{log(x_0/x)/log(a)}}$$

$$= e^{ln(th_0) 2^{log_a(x_0/x)}}$$

$$= th_0^{(x_0/x)^{1/log_2 a}}$$

Before continuing we want to plot some examples just to verify that they have the appropriate form.

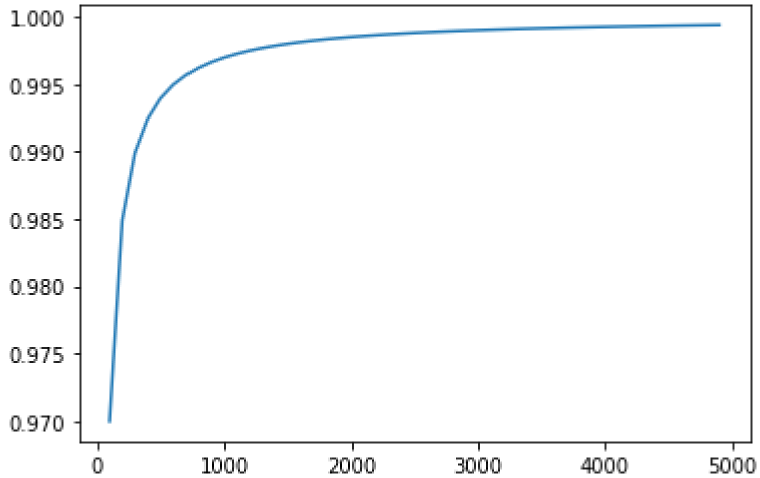
In the following figure, we plot this function using $a = 2, th_0 = 0.97, x_0 = 100$

In [40]:
```
def thresholdLevel(x, a, th0=0.97, x0=100): return (th0)**((x0/x)**(1/math.log2(a)))

hl1 = np.arange(100, 5000, 100)

plt.plot(hl1, thresholdLevel(hl1, 2))
```

Out[40]:
```
[<matplotlib.lines.Line2D at 0x1bbf89d55c0>]
```



In previous simulations, an empirical function was being used. That function was:

```
def thresholdLevel(th, hl, minhl=40): return (1 - th)*10/(hl - minhl)
```

if we plot both funcions for $a = 2, th_0 = 0.97, x_0 = 100$ we can see that both functions behave in the same way:

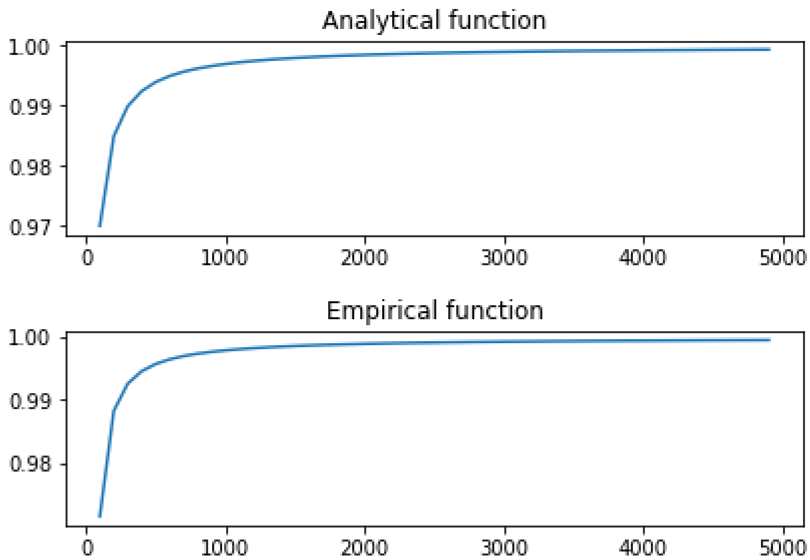
In [49]:
```
def empiricalThresholdLevel(hl, th=0.80, minhl=30): return 1 - (1 - th)*10/(hl - minh
l)

print (empiricalThresholdLevel(100))

hl1 = np.arange(100, 5000, 100)

plt.figure()
plt.subplot(211)
plt.plot(hl1, thresholdLevel(hl1, 2))
plt.title('Analytical function')
plt.subplot(212)
plt.plot(hl1, empiricalThresholdLevel(hl1))
plt.title('Empirical function')
# Adjust the subplot layout
plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.50, wspace
=0.35)
plt.show()
```

```
0.9714285714285714
```



Basic QPQ algorithm

- 1: Estimate the preference $\theta_i(r)$
- 2: Calculate the normalized preference $\bar{\theta}_i(r) = PIT(\theta_i(r))$
- 3: Declare a value $\theta_i(r)$ that represents her normalized preference.
- 4: Wait to receive the published normalized preferences $\dot{\theta}_j(r)$ from all players
- 5: For all $j \in N$ do
  - 6: if not GoF_Test($\theta_j(r), Historic_j$) then
    - 7: $\ddot{\theta}_j(r) \leftarrow \hat{\theta}_j(r)$, where $\hat{\theta}_j(r)$:= Pseudorandom($\theta_j(r)$)
  - 8: else
    - 9: $\ddot{\theta}_j(r) \leftarrow \dot{\theta}_j(r)$
  - 10: end if
  - 11: $Historic_j \leftarrow Historic_j \cup \{\theta_j(r)\}$
- 12: end for
- 13: Let $d = argmax_{j \in N}\{\theta_j(r)\}$
- 14: if $d = i$ then
  - 15: Player i gets resource r
- 16: end if

In [42]:

```python
# Values is a matrix (N x r) or (number of playes x number of rounds)
def QPQ(declaredVals, trueVals, histlen, maxpval, KSTest=True, debug=True):
    dims = declaredVals.shape
    N = dims[0] # number of players
    R = dims[1] # number of rounds
    decisions = np.full(R, np.nan)
    historic = np.full((N, histlen), np.nan) # Empty history matrix (N x histlen) or (number of playes x History Len)

    utilities = np.zeros((N, R)) # Empty utility matrix (N x R)
    falsenegatives = np.zeros((N, R)) # Empty utility matrix (N x R)
    for i in range(R):
        # Roll historic to the left
        if (i > histlen):
            historic = np.roll(historic, -1, axis=1)

        theta = np.zeros(N)
        # copy declared values at the end of the historic
        historic[:, min(i, histlen - 1)] = declaredVals[: , i]

        for j in range(N):
            if debug:
                print ("player ", j, " has values ", historic[j])
            if KSTest and stats.kstest(historic[j, 0:min(i+1, histlen)], 'uniform').pvalue < (1 - maxpval):
                if debug: print("False negative")
                theta[j] = np.random.uniform(0, 1, 1)
                falsenegatives[j, i] = 1
            else:
                theta[j] = declaredVals[j, i]

        # copy declared values at the end of the historic
        historic[:, min(i, histlen - 1)] = theta

        d = int(np.argmin(theta))
        if debug: print("Win player ", d)
        decisions[i] = d
        utilities[d, i] = trueVals[d, i]
    return decisions, utilities, falsenegatives
```

In [43]:
```
numplayers = 4
numLiars = 1
rounds = 1000
historyLen = 100
numberSimulations = 10
threshold = 0.95
betaFactor = 1.5

util1F  = np.full((numberSimulations, numplayers), np.nan)
falseneg1F  = np.full((numberSimulations, numplayers), np.nan)
util1T  = np.full((numberSimulations, numplayers), np.nan)
falseneg1T  = np.full((numberSimulations, numplayers), np.nan)

for k in range(numberSimulations):
    trueVals = np.random.uniform(0, 1, (numplayers, rounds))
    declaredVals = np.array(trueVals, copy=True)

    norm = stats.distributions.beta(betaFactor, 1)
    declaredVals[0:numLiars, :] = norm.ppf(trueVals[0:numLiars, :])

    #h = sns.jointplot(trueVals[0,:], declaredVals[0,:], stat_func=None)
    #h.set_axis_labels('original (True)', 'transformed (Declared)', fontsize=16);
    #h = sns.jointplot(trueVals[1,:], declaredVals[1,:], stat_func=None)
    #h.set_axis_labels('original (True)', 'transformed (Declared)', fontsize=16);


    rst1F, ut1F, fn1F = QPQ(declaredVals, trueVals, historyLen, threshold, KSTest=Fal
se, debug=False)
    rst1T, ut1T, fn1T = QPQ(declaredVals, trueVals, historyLen, threshold, KSTest=Tru
e, debug=False)


    util1F[k, :] = ut1F.mean(axis = 1)
    falseneg1F[k, :] = fn1F.mean(axis = 1)
    util1T[k, :] = ut1T.mean(axis = 1)
    falseneg1T[k, :] = fn1T.mean(axis = 1)

print("QPQ Player Utility without KS", util1F.mean(axis = 0))
print("QPQ Player Utility with KS", util1T.mean(axis = 0))
print("QPQ False Negative per player without KS", falseneg1F.mean(axis = 0))
print("QPQ False Negative per player with KS", falseneg1T.mean(axis = 0))
```

```
QPQ Player Utility without KS [0.02512731 0.05943611 0.06187023 0.05846252]
QPQ Player Utility with KS [0.05477253 0.05790794 0.05836647 0.05613937]
QPQ False Negative per player without KS [0. 0. 0. 0.]
QPQ False Negative per player with KS [0.2479 0.0201 0.0152 0.0192]
```

Multilevel QPQ algorithm

TODO

In [44]:
```
# Values is a matrix (N x r) or (number of playes x number of rounds)
# Level 0 is base level
# Level 1 is cluster level
def QPQ2Level(declaredVals, trueVals, K, histlenL0, maxpvalL0, histlenL1, maxpvalL1,
KSTest=True, debug=True):
    dims = declaredVals.shape
    N = dims[0] # number of players
    R = dims[1] # number of rounds
    playersPerCluster = int(N/K)
    if debug:
        print("Number of players ", N)
        print("Number of clusters ", K)
        print("Number of players per Cluster ", playersPerCluster)
    decisions = np.full(R, np.nan)
    historicL0 = np.full((N, histlenL0), np.nan) # Empty history matrix (N x histlen)
or (number of playes x History Len)
    historicL1 = np.full((K, histlenL1), np.nan)
    utilities = np.zeros((N, R)) # Empty utility matrix (N x R)
    falsenegativesL0 = np.zeros((N, R)) # Empty utility matrix (N x R)
    falsenegativesL1 = np.zeros((K, R)) # Empty utility matrix (K x R)
    falsenegativesBoth = np.zeros((N, R)) # Empty utility matrix (N x R)
    for i in range(R):
        # Roll historic to the left
        if (i > histlenL0):
            historicL0 = np.roll(historicL0, -1, axis=1)

        theta = np.zeros(N)
        # copy declared values at the end of the historic
        historicL0[:, min(i, histlenL0 - 1)] = declaredVals[: , i]

        for j in range(N):
            if debug:
                print ("player ", j, " has values ", historicL0[j])
            if KSTest and stats.kstest(historicL0[j, 0:min(i+1, histlenL0)], 'unifor
m').pvalue < (1 - maxpvalL0):
                if debug: print("False negative")
                theta[j] = np.random.uniform(0, 1, 1)
                falsenegativesL0[j, i] = 1
            else:
                theta[j] = declaredVals[j, i]

        # copy declared values at the end of the historic
        historicL0[:, min(i, histlenL0 - 1)] = theta

        thetaUp = np.zeros(K)
        decisionsL1 = np.full(K, np.nan)
        for k in range(K):
            valsCluster = theta[(k)*playersPerCluster:(k+1)*playersPerCluster]
            decisionsL1[k] = int(np.argmin(valsCluster))
            player = int((k)*playersPerCluster + decisionsL1[k])
            if debug:
                print ("cluster ", k, " has values ", valsCluster)
                print ("cluster ", k, " has player ", player)
                print ("cluster ", k, " has value ", valsCluster[int(decisionsL1[k
])])
                print ("cluster ", k, " has range ", (k, min(i, histlenL1 - 1)))
            historicL1[k, min(i, histlenL1 - 1)] = valsCluster[int(decisionsL1[k])]
            if KSTest and stats.kstest(historicL1[k, 0:min(i+1, histlenL1)], stats.be
ta(a=1., b=playersPerCluster).cdf).pvalue < (1 - maxpvalL1):
                if debug: print("False negative")
                thetaUp[k] = np.random.uniform(0, 1, 1)
                falsenegativesL1[k, i] = 1
```

```
                falsenegativesBoth[player, i] = 1
            else:
                thetaUp[k] = valsCluster[int(decisionsL1[k])]

        #print(thetaUp)
        dCluster = np.argmin(thetaUp)
        d = int(dCluster*playersPerCluster + decisionsL1[dCluster])
        if debug: print("Win cluster ", dCluster, " and player ", d)
        decisions[i] = d
        utilities[d, i] = trueVals[d, i]
    return decisions, utilities, falsenegativesL0, falsenegativesL1, falsenegativesBo
th
```

In [45]:
```
numplayers = 4
numclusters = 2
numLiars = 1
rounds = 1000
historyLen = 100
alpha = 1
hlL0 = int((historyLen * numplayers) / (numclusters + alpha * numplayers / numcluster
s))
hlL1 = int(alpha * hlL0)

# memoria disponible = hlL0 * numclusters +  hlL1 * (numplayers / numclusters) = hist
oryLen * numplayers
# suponiendo hlL1 = alpha * hlL0 entonces:
#   hlL0 = (historyLen * numplayers) / (numclusters + alpha * numplayers / numcluster
s)
# y alpha =  hlL1 / hlL0

# falso negativo = 1 - (1 -  pvc)*(1 - pvu) <-- independiente de la historyLen
# pv = 1 - (1 -  pvc)(1 - pvu) ==>  pvu = 1 - (1 - pv)/(1 -  pvc)
# pvc = pvu = 1 - sqrt(1 - pv) = cte1 / hlL1 = cte0 / hlL0
# negativo cierto en beta SI depende de la historia

numberSimulations = 10
thresholdL0 = 0.98
thresholdL1 = 0.95
betaFactor = 1.5

util2F  = np.full((numberSimulations, numplayers), np.nan)
falseneg2F  = np.full((numberSimulations, numplayers), np.nan)
util2T  = np.full((numberSimulations, numplayers), np.nan)
falseneg2T  = np.full((numberSimulations, numplayers), np.nan)

for k in range(numberSimulations):
    trueVals = np.random.uniform(0, 1, (numplayers, rounds))
    declaredVals = np.array(trueVals, copy=True)

    norm = stats.distributions.beta(betaFactor, 1)
    declaredVals[0:numLiars, :] = norm.ppf(trueVals[0:numLiars, :])

    rst2F, ut2F, fn2F, fnc2F, fnb2F = QPQ2Level(declaredVals, trueVals, numclusters,
hlL0, thresholdL0, hlL1, thresholdL1, KSTest=False, debug=False)
    rst2T, ut2T, fn2T, fnc2T, fnb2T = QPQ2Level(declaredVals, trueVals, numclusters,
hlL0, thresholdL0, hlL1, thresholdL1, KSTest=True, debug=False)


    util2F[k, :] = ut2F.mean(axis = 1)
    falseneg2F[k, :] = fn2F.mean(axis = 1)
    util2T[k, :] = ut2T.mean(axis = 1)
    falseneg2T[k, :] = fn2T.mean(axis = 1)

print("QPQ2 Player Utility without KS", util2F.mean(axis = 0))
print("QPQ2 Player Utility with KS", util2T.mean(axis = 0))
print("QPQ2 False Negative per player without KS", falseneg2F.mean(axis = 0))
print("QPQ2 False Negative per player with KS", falseneg2T.mean(axis = 0))
```

```
QPQ2 Player Utility without KS [0.02486683 0.05882594 0.06320268 0.05974674]
QPQ2 Player Utility with KS [0.05073252 0.05841636 0.06326611 0.06213848]
QPQ2 False Negative per player without KS [0. 0. 0. 0.]
QPQ2 False Negative per player with KS [0.2157 0.0071 0.0086 0.0052]
```

Comparative analysis

TODO Add description

In [47]:
```
# Number of simulations
numberSimulation = 200

# History length
historyLenArray = [100]
alpha = 1

# Number of rounds
rounds = 1000

# Number of players
numplayersArray = [16, 32]
# numplayersArray = [8, 16, 32, 64, 128, 256, 512]

# Number of clusters
numclusters = 8

# Threshold
thresholdBase = 0.80
minhl = 30

def thresholdLevel(x, a, th0=0.97, x0=100): return (th0)**((x0/x)**(1/math.log2(a)))
def empiricalThresholdLevel(hl, th=0.82, minhl=40): return 1 - (1 - th)*10/(hl - minh
l)

def doSimulation(numplayers, numLiars, numclusters, thresholdFunct):
    historyLen = historyLenArray[0]
    print("Simulation using numplayers =", numplayers, " and numclusters = ", numclus
ters)
    results1F   = np.full((numberSimulation, numplayers), np.nan)
    util1F  = np.full((numberSimulation, numplayers), np.nan)
    falseneg1F  = np.full((numberSimulation, numplayers), np.nan)
    results1T   = np.full((numberSimulation, numplayers), np.nan)
    util1T  = np.full((numberSimulation, numplayers), np.nan)
    falseneg1T  = np.full((numberSimulation, numplayers), np.nan)


    results2F   = np.full((numberSimulation, numplayers), np.nan)
    util2F  = np.full((numberSimulation, numplayers), np.nan)
    falseneg2F = np.full((numberSimulation, numplayers), np.nan)
    results2T   = np.full((numberSimulation, numplayers), np.nan)
    util2T  = np.full((numberSimulation, numplayers), np.nan)
    falseneg2T = np.full((numberSimulation, numplayers), np.nan)

    # old function threshold = empiricalThresholdLevel(historyLen)
    threshold = thresholdFunct(historyLen)

    hlL0 = int((historyLen * numplayers) / (numclusters + alpha * numplayers / numclu
sters))
    hlL1 = int(alpha * hlL0)

    thresholdL0 = thresholdFunct(hlL0)
    thresholdL1 = thresholdFunct(hlL1)
    print("    Using historyLen at QPQ =", historyLen)
    print("    Using historyLen L0 at QPQ2 =", hlL0, " and historyLen L1 at QPQ2 =",
hlL1)
    print("    Using threshold at QPQ =", threshold)
    print("    Using threshold L0 at QPQ2 =", thresholdL0, " and threshold L1 at QPQ2
=", thresholdL1)
    for k in range(numberSimulation):
        # print("Simulation number=",k)
        trueVals = np.random.uniform(0, 1, (numplayers, rounds))
```

```
        declaredVals = np.array(trueVals, copy=True)
        if (numLiars > 0):
            norm = stats.distributions.beta(betaFactor, 1)
            declaredVals[0:numLiars, :] = norm.ppf(trueVals[0:numLiars, :])

        rst1F, ut1F, fn1F                  = QPQ(declaredVals, trueVals, historyLen, th
reshold, KSTest=False, debug=False)
        rst2F, ut2F, fn2F, fnc2F, fnb2F   = QPQ2Level(declaredVals, trueVals, numclust
ers, hlL0, thresholdL0, hlL1, thresholdL1, KSTest=False, debug=False)
        rst1T, ut1T, fn1T                  = QPQ(declaredVals, trueVals, historyLen, th
reshold, KSTest=True, debug=False)
        rst2T, ut2T, fn2T, fnc2T, fnb2T   = QPQ2Level(declaredVals, trueVals, numclust
ers, hlL0, thresholdL0, hlL1, thresholdL1, KSTest=True, debug=False)

        util1F[k, :] = ut1F.mean(axis = 1)
        falseneg1F[k, :] = fn1F.mean(axis = 1)
        util1T[k, :] = ut1T.mean(axis = 1)
        falseneg1T[k, :] = fn1T.mean(axis = 1)

        util2F[k, :] = ut2F.mean(axis = 1)
        falseneg2F[k, :] = fnb2F.mean(axis = 1)
        util2T[k, :] = ut2T.mean(axis = 1)
        falseneg2T[k, :] = fnb2T.mean(axis = 1)

    print("    QPQ Player Utility without KS", util1F.mean(axis = 0))
    print("    QPQ Player Utility with KS", util1T.mean(axis = 0))
    print("    QPQ False Negative per player without KS", falseneg1F.mean(axis = 0))
    print("    QPQ False Negative per player with KS", falseneg1T.mean(axis = 0))
    print("    QPQ2 Player Utility without KS", util2F.mean(axis = 0))
    print("    QPQ2 Player Utility with KS", util2T.mean(axis = 0))
    print("    QPQ2 False Negative per player without KS", falseneg2F.mean(axis = 0))
    print("    QPQ2 False Negative per player with KS", falseneg2T.mean(axis = 0))
    if (numLiars > 0):
        print("    QPQ Player Utility. Honests:", util1T[numLiars:].mean(), " Dishone
sts:", util1T[0:numLiars].mean())
        print("    QPQ2 Player Utility. Honests:", util2T[numLiars:].mean(), " Dishon
ests:", util2T[0:numLiars].mean())
        print("    QPQ False Negative per player. Honests:", falseneg1T[numLiars:].me
an(), " Dishonests:", falseneg1T[0:numLiars].mean())
        print("    QPQ2 False Negative per player. Honests:", falseneg2T[numLiars:].m
ean(), " Dishonests:", falseneg2T[0:numLiars].mean())
        return util1T[numLiars:].mean()/util2T[numLiars:].mean(), util1T[0:numLiars].
mean()/util2T[0:numLiars].mean()
    else:
        print("    QPQ Player Utility ", util1T.mean())
        print("    QPQ2 Player Utility ", util2T.mean())
        print("    QPQ False Negative per player ", falseneg1T.mean())
        print("    QPQ2 False Negative per player ", falseneg2T.mean())
        return util1T.mean()/util2T.mean(), 1


ratiosHonest0 = np.full(len(numplayersArray), np.nan)
ratiosDishonest0 = np.full(len(numplayersArray), np.nan)
ratiosHonest1 = np.full(len(numplayersArray), np.nan)
ratiosDishonest1 = np.full(len(numplayersArray), np.nan)

# for idx1, historyLen in enumerate(historyLenArray):
for idx, numplayers in enumerate(numplayersArray):
    a = numplayers * numclusters / ( numclusters ^ 2 + numplayers )
    tfold = lambda hl: empiricalThresholdLevel(hl)
    tfnew = lambda hl: thresholdLevel(hl, a)

    ratiosHonest0[idx], ratiosDishonest0[idx] = doSimulation(numplayers, 0, numcluste
```

```
rs, tfold)
    print("Player Utility Ratio QPQ/QPQ2. Old Function. All honest players:", ratiosH
onest0[idx])

    ratiosHonest1[idx], ratiosDishonest1[idx] = doSimulation(numplayers, 1, numcluste
rs, tfold)
    print("Player Utility Ratio QPQ/QPQ2. Old Function. Honest players:", ratiosHones
t1[idx], " Dishonest player:", ratiosDishonest1[idx])

    ratiosHonest0[idx], ratiosDishonest0[idx] = doSimulation(numplayers, 0, numcluste
rs, tfnew)
    print("Player Utility Ratio QPQ/QPQ2. New Function. All honest players:", ratiosH
onest0[idx])

    ratiosHonest1[idx], ratiosDishonest1[idx] = doSimulation(numplayers, 1, numcluste
rs, tfnew)
    print("Player Utility Ratio QPQ/QPQ2. New Function. Honest players:", ratiosHones
t1[idx], " Dishonest player:", ratiosDishonest1[idx])
```

```
Simulation using numplayers = 16  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 160  and historyLen L1 at QPQ2 = 160
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.985  and threshold L1 at QPQ2 = 0.985
    QPQ Player Utility without KS [0.00374634 0.00363387 0.00370569 0.00357138 0.003
67935 0.00367055
 0.00366582 0.00364617 0.00365841 0.00366599 0.00371054 0.00372734
 0.00368082 0.00367166 0.00377587 0.00371437]
    QPQ Player Utility with KS [0.00407155 0.00397556 0.00411946 0.00394638 0.004041
38 0.0039631
 0.00406624 0.00395117 0.00402914 0.00392035 0.00404695 0.00405806
 0.00398093 0.00405263 0.00404824 0.0041486 ]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0.]
    QPQ False Negative per player with KS [0.011955 0.012885 0.01238  0.011675 0.012
425 0.01185  0.012775 0.013175
 0.012695 0.010185 0.013335 0.01099  0.01118  0.01235  0.01195  0.01299 ]
    QPQ2 Player Utility without KS [0.00374634 0.00363387 0.00370569 0.00357138 0.00
367935 0.00367055
 0.00366582 0.00364617 0.00365841 0.00366599 0.00371054 0.00372734
 0.00368082 0.00367166 0.00377587 0.00371437]
    QPQ2 Player Utility with KS [0.00407877 0.00393686 0.00397312 0.00383181 0.00388
663 0.00390105
 0.00388235 0.00392928 0.00388682 0.00387318 0.00404597 0.00402899
 0.00396599 0.0040001  0.00401048 0.00406056]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.005775 0.005815 0.003235 0.00347  0.00
3125 0.00309  0.003485 0.00361
 0.00517  0.00487  0.00717  0.007425 0.007245 0.007465 0.00385  0.00376 ]
    QPQ Player Utility  0.004026233911308328
    QPQ2 Player Utility  0.0039557464111865635
    QPQ False Negative per player  0.012174687500000001
    QPQ2 False Negative per player  0.00491
Player Utility Ratio QPQ/QPQ2. Old Function. All honest players: 1.0178190138585301
Simulation using numplayers = 16  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 160  and historyLen L1 at QPQ2 = 160
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.985  and threshold L1 at QPQ2 = 0.985
    QPQ Player Utility without KS [0.0005928  0.00393998 0.00397559 0.00398233 0.003
93544 0.00396101
 0.00393821 0.00401111 0.00402059 0.00399255 0.00393693 0.00406365
 0.00396106 0.00395689 0.00399231 0.00393081]
    QPQ Player Utility with KS [0.0089943  0.0041704  0.00427843 0.00424652 0.004180
61 0.00421345
 0.00415118 0.00424625 0.00428388 0.00421711 0.00426631 0.00432971
 0.00430569 0.00417487 0.00424248 0.00426483]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0.]
    QPQ False Negative per player with KS [0.231515 0.012625 0.012915 0.01189  0.011
84  0.011335 0.010095 0.01142
 0.013385 0.011455 0.01281  0.01328  0.012965 0.011595 0.01316  0.01346 ]
    QPQ2 Player Utility without KS [0.0005928  0.00393998 0.00397559 0.00398233 0.00
393544 0.00396101
 0.00393821 0.00401111 0.00402059 0.00399255 0.00393693 0.00406365
 0.00396106 0.00395689 0.00399231 0.00393081]
    QPQ2 Player Utility with KS [0.01041216 0.00660396 0.00424972 0.00430306 0.00432
206 0.00436333
 0.00411873 0.00419705 0.00425062 0.00422315 0.00411909 0.00424277
 0.00417758 0.00422116 0.0043096  0.00414798]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.095425 0.115865 0.012985 0.012605 0.01
444  0.014105 0.00612  0.00608
 0.008385 0.008435 0.002035 0.002175 0.0046   0.005185 0.008045 0.00781 ]
    QPQ Player Utility. Honests: 0.004535694990270138  Dishonests: 0.004471997933270
697
    QPQ2 Player Utility. Honests: 0.004768171633761016  Dishonests: 0.00440901994564
7522
    QPQ False Negative per player. Honests: 0.025970477386934674  Dishonests: 0.0286
87499999999998
    QPQ2 False Negative per player. Honests: 0.020370288944723623  Dishonests: 0.0
Player Utility Ratio QPQ/QPQ2. Old Function. Honest players: 0.9512440697719797  Dis
honest player: 1.014283897192469
Simulation using numplayers = 16  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 160  and historyLen L1 at QPQ2 = 160
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9754770064355601  and threshold L1 at QPQ2 = 0.97
54770064355601
    QPQ Player Utility without KS [0.00368237 0.00371961 0.00367316 0.00364707 0.003
62942 0.00363337
 0.00363887 0.00364557 0.00368288 0.00369109 0.00370361 0.00363117
 0.00363384 0.00375194 0.00364444 0.00369673]
    QPQ Player Utility with KS [0.00412298 0.00401059 0.0039715  0.00393195 0.003970
55 0.00390114
 0.00388794 0.00395694 0.00398255 0.00404496 0.00404442 0.00394773
 0.00397452 0.00411498 0.00397122 0.0040842 ]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0.]
    QPQ False Negative per player with KS [0.011425 0.010715 0.010635 0.01063  0.012
595 0.010845 0.00986  0.01172
 0.011995 0.013815 0.012695 0.01184  0.013465 0.01166  0.01168  0.01294 ]
    QPQ2 Player Utility without KS [0.00368237 0.00371961 0.00367316 0.00364707 0.00
362942 0.00363337
 0.00363887 0.00364557 0.00368288 0.00369109 0.00370361 0.00363117
 0.00363384 0.00375194 0.00364444 0.00369673]
    QPQ2 Player Utility with KS [0.00414476 0.00416537 0.00392179 0.00398635 0.00416
227 0.00412997
 0.0041519  0.00414396 0.00412111 0.00425319 0.00425857 0.0041623
 0.00423298 0.00443456 0.00406414 0.00406323]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.00474  0.00426  0.00257  0.002915 0.01
4515 0.014535 0.010035 0.01006
 0.01292  0.012405 0.012065 0.01258  0.016875 0.01699  0.00557  0.005365]
    QPQ Player Utility  0.003994886173141917
    QPQ2 Player Utility  0.004149778377474532
    QPQ False Negative per player  0.011782187500000001
    QPQ2 False Negative per player  0.009900000000000003
Player Utility Ratio QPQ/QPQ2. New Function. All honest players: 0.9626745839793788
Simulation using numplayers = 16  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 160  and historyLen L1 at QPQ2 = 160
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9754770064355601  and threshold L1 at QPQ2 = 0.97
54770064355601
    QPQ Player Utility without KS [0.00061332 0.00398256 0.00395472 0.00397983 0.004
00057 0.0039769
 0.00398935 0.00395758 0.0040015  0.00389804 0.00391199 0.00398506
 0.0039835  0.00396112 0.0040678  0.00398374]
    QPQ Player Utility with KS [0.00877516 0.004176   0.00413796 0.0041774  0.004203
94 0.0042035
 0.00421757 0.00416356 0.00419013 0.00418244 0.00411611 0.00421049
```

```
 0.00426035 0.0042243  0.00427924 0.00429084]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0.]
    QPQ False Negative per player with KS [0.23371  0.01094  0.009605 0.011685 0.009
26  0.012235 0.011085 0.010355
 0.010665 0.01308  0.00922  0.01127  0.012475 0.01095  0.01185  0.01307 ]
    QPQ2 Player Utility without KS [0.00061332 0.00398256 0.00395472 0.00397983 0.00
400057 0.0039769
 0.00398935 0.00395758 0.0040015  0.00389804 0.00391199 0.00398506
 0.0039835  0.00396112 0.0040678  0.00398374]
    QPQ2 Player Utility with KS [0.0111598  0.00653911 0.00420123 0.00435103 0.00428
6   0.00428487
 0.00428544 0.00421804 0.00434346 0.00429492 0.00420968 0.00418386
 0.0042806  0.0042558  0.00441065 0.00429739]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.095125 0.11528  0.006235 0.006075 0.00
2525 0.00267  0.010605 0.01047
 0.01044  0.0102   0.006425 0.00643  0.004285 0.00416  0.005305 0.00512 ]
    QPQ Player Utility. Honests: 0.004485713751113851  Dishonests: 0.004955583568276
926
    QPQ2 Player Utility. Honests: 0.004847522146912459  Dishonests: 0.00536661411707
49405
    QPQ False Negative per player. Honests: 0.02508071608040201  Dishonests: 0.02712
5000000000003
    QPQ2 False Negative per player. Honests: 0.018655150753768845  Dishonests: 0.054
5
Player Utility Ratio QPQ/QPQ2. New Function. Honest players: 0.9253621984937078  Dis
honest player: 0.9234097067850957
Simulation using numplayers = 32  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 266  and historyLen L1 at QPQ2 = 266
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9920353982300885  and threshold L1 at QPQ2 = 0.99
20353982300885
    QPQ Player Utility without KS [0.00092809 0.000976   0.00097143 0.00093525 0.000
96132 0.00096098
 0.00092945 0.00093426 0.00093765 0.00094891 0.00097028 0.00095592
 0.00094078 0.00093366 0.00097888 0.00094072 0.00093075 0.00093052
 0.00094932 0.00094764 0.00092598 0.00094875 0.00094978 0.00095346
 0.0009412  0.00094101 0.00095477 0.00095509 0.00095666 0.00097504
 0.00093494 0.00094032]
    QPQ Player Utility with KS [0.00117787 0.00108851 0.00115158 0.00116402 0.001134
43 0.00117385
 0.0010996  0.00107451 0.00110741 0.00114788 0.00115747 0.00113566
 0.00107771 0.00108721 0.00112593 0.00111032 0.00110931 0.00114015
 0.00113378 0.00111969 0.00110599 0.00113612 0.00112955 0.00115841
 0.00108173 0.00108594 0.00109411 0.00112393 0.00113025 0.00121008
 0.00107966 0.00111087]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ False Negative per player with KS [0.01388  0.00932  0.013105 0.01313  0.012
815 0.01217  0.012045 0.01173
 0.011975 0.011935 0.01197  0.01146  0.010255 0.011875 0.011105 0.012965
 0.01456  0.013415 0.011585 0.01291  0.0144   0.01135  0.01137  0.012635
 0.011075 0.01013  0.01116  0.010885 0.014315 0.01351  0.010215 0.010915]
    QPQ2 Player Utility without KS [0.00092809 0.000976   0.00097143 0.00093525 0.00
096132 0.00096098
 0.00092945 0.00093426 0.00093765 0.00094891 0.00097028 0.00095592
 0.00094078 0.00093366 0.00097888 0.00094072 0.00093075 0.00093052
 0.00094932 0.00094764 0.00092598 0.00094875 0.00094978 0.00095346
 0.0009412  0.00094101 0.00095477 0.00095509 0.00095666 0.00097504
```

```
 0.00093494 0.00094032]
    QPQ2 Player Utility with KS [0.00096931 0.00102015 0.00104438 0.00098565 0.00100
652 0.00103403
 0.00095256 0.00095981 0.0009836  0.00099971 0.00102812 0.00098873
 0.00097012 0.00097943 0.00100737 0.00099215 0.00101633 0.00098981
 0.00101195 0.0010092  0.00102459 0.00100828 0.00101173 0.00101287
 0.00098867 0.00100862 0.00098675 0.00099807 0.00100867 0.00102261
 0.00096039 0.00101199]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.0026   0.002455 0.00242  0.002545 0.00
017  0.000135 0.000125 0.00014
 0.001245 0.001235 0.001315 0.001335 0.000275 0.00026  0.000285 0.000285
 0.00299  0.002815 0.00292  0.003125 0.003405 0.003615 0.003535 0.003595
 0.000915 0.000855 0.000825 0.000905 0.001445 0.00133  0.00146  0.00132 ]
    QPQ Player Utility  0.0011238601407196581
    QPQ2 Player Utility  0.0009997560786156332
    QPQ False Negative per player  0.012067656250000001
    QPQ2 False Negative per player  0.0016212500000000003
Player Utility Ratio QPQ/QPQ2. Old Function. All honest players: 1.1241343411243594
Simulation using numplayers = 32  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 266  and historyLen L1 at QPQ2 = 266
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9920353982300885  and threshold L1 at QPQ2 = 0.99
20353982300885
    QPQ Player Utility without KS [8.67131080e-05 9.71161378e-04 9.89779899e-04 1.00
586605e-03
 9.58362252e-04 1.00111490e-03 9.96005040e-04 1.01908966e-03
 9.60018986e-04 9.73471107e-04 1.03590134e-03 9.96660430e-04
 9.84890582e-04 9.80548267e-04 9.89795546e-04 9.95894123e-04
 1.00042534e-03 1.02600306e-03 1.01218890e-03 9.62727061e-04
 1.02842763e-03 1.00951127e-03 9.98445358e-04 1.01822902e-03
 9.70210394e-04 9.55270606e-04 1.02436512e-03 9.89206827e-04
 9.80871414e-04 9.78707661e-04 9.73860887e-04 9.85312871e-04]
    QPQ Player Utility with KS [0.0042561  0.00115486 0.00110853 0.00117083 0.001103
39 0.00122067
 0.00118154 0.00118505 0.00112491 0.00111174 0.0012083  0.00116904
 0.00121439 0.00112215 0.00117629 0.00113767 0.00119684 0.00123892
 0.00113526 0.0011389  0.00121244 0.00112133 0.00119042 0.00119332
 0.00112274 0.00119016 0.00120556 0.00112539 0.00116494 0.00110813
 0.00108676 0.00108377]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ False Negative per player with KS [0.23485  0.013295 0.01059  0.01261  0.012
385 0.01072  0.014055 0.01319
 0.01081  0.010835 0.011565 0.01277  0.012565 0.01055  0.012655 0.011675
 0.012355 0.014005 0.01151  0.01292  0.0139   0.01098  0.0124   0.01187
 0.009335 0.013255 0.011815 0.011105 0.01188  0.0108   0.01167  0.00976 ]
    QPQ2 Player Utility without KS [8.67131080e-05 9.71161378e-04 9.89779899e-04 1.0
0586605e-03
 9.58362252e-04 1.00111490e-03 9.96005040e-04 1.01908966e-03
 9.60018986e-04 9.73471107e-04 1.03590134e-03 9.96660430e-04
 9.84890582e-04 9.80548267e-04 9.89795546e-04 9.95894123e-04
 1.00042534e-03 1.02600306e-03 1.01218890e-03 9.62727061e-04
 1.02842763e-03 1.00951127e-03 9.98445358e-04 1.01822902e-03
 9.70210394e-04 9.55270606e-04 1.02436512e-03 9.89206827e-04
 9.80871414e-04 9.78707661e-04 9.73860887e-04 9.85312871e-04]
    QPQ2 Player Utility with KS [0.00547716 0.00110091 0.00107756 0.00116041 0.00097
92  0.00103333
 0.00102112 0.00108224 0.00102076 0.00100063 0.00107553 0.00107707
```

```
 0.00104295 0.00104746 0.00105679 0.0010247  0.00104062 0.0010914
 0.00106253 0.00099056 0.00111387 0.00105354 0.00104662 0.00113525
 0.00104457 0.00098902 0.00106352 0.00103333 0.0010508  0.00103755
 0.00099536 0.00101871]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.012605 0.016455 0.016635 0.01715  0.00
0465 0.00043  0.000345 0.00039
 0.001225 0.00112  0.001345 0.001285 0.002275 0.00224  0.00236  0.002245
 0.001045 0.001225 0.00108  0.00089  0.001845 0.00199  0.00182  0.001955
 0.00081  0.000695 0.00072  0.000805 0.00347  0.00332  0.0033   0.0033  ]
    QPQ Player Utility. Honests: 0.001254438830036878  Dishonests: 0.001368867943512
5684
    QPQ2 Player Utility. Honests: 0.0011894794190925929  Dishonests: 0.0010754253650
073799
    QPQ False Negative per player. Honests: 0.018882537688442214  Dishonests: 0.0216
25000000000002
    QPQ2 False Negative per player. Honests: 0.0033525439698492464  Dishonests: 0.00
059375
Player Utility Ratio QPQ/QPQ2. Old Function. Honest players: 1.0546116308543112  Dis
honest player: 1.2728618721980534
Simulation using numplayers = 32  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 266  and historyLen L1 at QPQ2 = 266
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9792868137050521  and threshold L1 at QPQ2 = 0.97
92868137050521
    QPQ Player Utility without KS [0.00092061 0.000955   0.00093033 0.00096455 0.000
96694 0.00094157
 0.00094922 0.00091    0.00093713 0.00092704 0.00093662 0.00097758
 0.00094288 0.00096027 0.00092732 0.00097136 0.00092673 0.00094787
 0.00093393 0.00095847 0.00095451 0.00096644 0.00095398 0.00094816
 0.00096007 0.0009414  0.00094598 0.00093416 0.00096974 0.000949
 0.00096286 0.00094008]
    QPQ Player Utility with KS [0.00110496 0.00113434 0.00113641 0.00115332 0.001154
41 0.0011155
 0.00112254 0.00114932 0.00105019 0.00106717 0.00109759 0.00116281
 0.00111113 0.00111432 0.00111306 0.00111551 0.00108639 0.00108232
 0.00110842 0.00114228 0.00110842 0.00110589 0.00113287 0.00112914
 0.00114838 0.00114617 0.00114696 0.00105979 0.00112197 0.00113096
 0.00113256 0.00109612]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ False Negative per player with KS [0.01122  0.011735 0.011465 0.01173  0.012
535 0.012145 0.01219  0.014415
 0.011085 0.011385 0.01074  0.011015 0.0115   0.0116   0.013415 0.012885
 0.01048  0.01048  0.0129   0.011375 0.01123  0.01216  0.013885 0.013005
 0.01045  0.013595 0.010065 0.01088  0.011885 0.01219  0.011065 0.01185 ]
    QPQ2 Player Utility without KS [0.00092061 0.000955   0.00093033 0.00096455 0.00
096694 0.00094157
 0.00094922 0.00091    0.00093713 0.00092704 0.00093662 0.00097758
 0.00094288 0.00096027 0.00092732 0.00097136 0.00092673 0.00094787
 0.00093393 0.00095847 0.00095451 0.00096644 0.00095398 0.00094816
 0.00096007 0.0009414  0.00094598 0.00093416 0.00096974 0.000949
 0.00096286 0.00094008]
    QPQ2 Player Utility with KS [0.00103808 0.00105488 0.00110673 0.0010761  0.00106
458 0.00109684
 0.00106848 0.00106192 0.00105867 0.00101387 0.00108294 0.00112771
 0.00107057 0.00108113 0.00110893 0.00107981 0.00106428 0.00109177
 0.00113402 0.00109984 0.00112054 0.00108533 0.00110357 0.00107069
 0.00110526 0.00107208 0.00107321 0.00103353 0.00107878 0.00107795
```

```
 0.00112144 0.00104733]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ2 False Negative per player with KS [0.00717  0.007635 0.00769  0.00742  0.00
155  0.00141  0.00155  0.00158
 0.00402  0.00428  0.00409  0.00435  0.00286  0.00294  0.00269  0.002815
 0.0058   0.005585 0.005525 0.005485 0.00249  0.00252  0.00271  0.00245
 0.00616  0.006095 0.006185 0.005945 0.002915 0.00322  0.003125 0.003185]
    QPQ Player Utility  0.001118163437136685
    QPQ2 Player Utility  0.001080339391045631
    QPQ False Negative per player  0.01182984375
    QPQ2 False Negative per player  0.00417015625
Player Utility Ratio QPQ/QPQ2. New Function. All honest players: 1.0350112625759624
Simulation using numplayers = 32  and numclusters =  8
    Using historyLen at QPQ = 100
    Using historyLen L0 at QPQ2 = 266  and historyLen L1 at QPQ2 = 266
    Using threshold at QPQ = 0.97
    Using threshold L0 at QPQ2 = 0.9792868137050521  and threshold L1 at QPQ2 = 0.97
92868137050521
    QPQ Player Utility without KS [7.96122293e-05 9.78640117e-04 1.01229381e-03 9.69
072412e-04
 9.97933071e-04 1.02202502e-03 9.62662653e-04 1.00045791e-03
 9.93804007e-04 9.93227823e-04 9.88100907e-04 9.86538628e-04
 1.00397814e-03 9.97718595e-04 9.51793095e-04 1.00421092e-03
 9.88427668e-04 9.93473399e-04 9.76951730e-04 9.94849793e-04
 9.79212567e-04 9.80883904e-04 9.84335736e-04 9.81543233e-04
 9.86975486e-04 9.66982507e-04 9.71283973e-04 9.72438212e-04
 9.92677109e-04 9.78198055e-04 9.86806978e-04 1.00192800e-03]
    QPQ Player Utility with KS [0.00424778 0.00113842 0.00116361 0.00115378 0.001168
57 0.00122164
 0.00112801 0.0011366  0.00115147 0.00112751 0.00113451 0.00114315
 0.00121306 0.00110714 0.001088   0.00113093 0.00114474 0.00115964
 0.00114445 0.0011837  0.00123843 0.00117337 0.00116849 0.00118606
 0.0011315  0.00111726 0.00111851 0.00111691 0.00115564 0.00112337
 0.00114544 0.00114497]
    QPQ False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
    QPQ False Negative per player with KS [0.234725 0.012815 0.009995 0.01213  0.010
175 0.01222  0.012515 0.01173
 0.013    0.011975 0.011285 0.014155 0.01411  0.01176  0.01114  0.01017
 0.01185  0.011805 0.012855 0.01255  0.01395  0.01314  0.01134  0.012755
 0.010985 0.01089  0.012125 0.01159  0.011775 0.011755 0.010455 0.011135]
    QPQ2 Player Utility without KS [7.96122293e-05 9.78640117e-04 1.01229381e-03 9.6
9072412e-04
 9.97933071e-04 1.02202502e-03 9.62662653e-04 1.00045791e-03
 9.93804007e-04 9.93227823e-04 9.88100907e-04 9.86538628e-04
 1.00397814e-03 9.97718595e-04 9.51793095e-04 1.00421092e-03
 9.88427668e-04 9.93473399e-04 9.76951730e-04 9.94849793e-04
 9.79212567e-04 9.80883904e-04 9.84335736e-04 9.81543233e-04
 9.86975486e-04 9.66982507e-04 9.71283973e-04 9.72438212e-04
 9.92677109e-04 9.78198055e-04 9.86806978e-04 1.00192800e-03]
    QPQ2 Player Utility with KS [0.00586962 0.00127029 0.0012422  0.00125189 0.00112
72  0.00116038
 0.00110051 0.00114256 0.00112121 0.00107432 0.00110061 0.00112301
 0.00117793 0.00116846 0.00116046 0.00115259 0.00112708 0.00111367
 0.00124325 0.00112489 0.00111088 0.00110645 0.00112342 0.00118596
 0.0010938  0.00109239 0.00107514 0.00112026 0.00116074 0.0011054
 0.00110009 0.00114226]
    QPQ2 False Negative per player without KS [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
    QPQ2 False Negative per player with KS [0.02987  0.03923  0.03799  0.03803  0.00
588  0.006245 0.00601  0.00616
 0.005535 0.00547  0.00573  0.00542  0.007505 0.007535 0.00729  0.00758
 0.00263  0.002735 0.002565 0.002675 0.00217  0.00228  0.002325 0.002255
 0.004095 0.003745 0.00403  0.003795 0.0018   0.001605 0.001695 0.00186 ]
    QPQ Player Utility. Honests: 0.0012467143145405554  Dishonests: 0.00132037926767
13816
    QPQ2 Player Utility. Honests: 0.0012896188139687934  Dishonests: 0.0012967054204
675798
    QPQ False Negative per player. Honests: 0.018896042713567842  Dishonests: 0.0200
3125
    QPQ2 False Negative per player. Honests: 0.008275125628140704  Dishonests: 0.001
6250000000000001
Player Utility Ratio QPQ/QPQ2. New Function. Honest players: 0.9667308673202435  Dis
honest player: 1.0182569200607376
```

In [ ]: