Caitlyn Jones

Prof. Yangyang Wang

MATH 40A: Intro to Applied Math

8 December 2024

Lab 6

1. I modified `test_strategy.py` so that it wins more than 5% of the rounds.

```
def play(n,hist,lw,lc,y):

    #Return an offset number of the last winning number
    return (lw + 5) % 100
```

```
Wins for random_choice : 57.41% (1.20% per player)
Wins for test_strategy : 13.77%
No winner              : 28.82%
```

Instead of function `play()` returning 10, it returns a number that is offset from the last winning number in order to avoid collisions. I implemented another strategy as well.

```
def play(n,hist,lw,lc,y):

    #Returns numbers that are less frequently chosen
    return min(range(len(hist)), key=lambda x: hist[x])
```

```
Wins for random_choice : 60.02% (1.25% per player)
Wins for test_strategy : 39.98%
No winner              :  0.00%
```

In this one, `play()` is returning a number that is less frequently chosen in order to increase the chances of a unique guess.

2. My unique number strategy first creates an array filled with all the choices from the last round of playing. Then, I created a new array that holds the numbers that were not chosen in the last round. My function then either returns a random number from the unused numbers array or 0 if the array is empty. Below is the implementation.

```python
from random import choice

'''
This function first creates an array based on
the numbers that were chosen in the last round.
Then, it makes an array based on numbers that
were NOT chosen. If this array has values, then
the function returns a random one from here, if
not, the function returns 0.
'''
def play(n,hist,lw,lc,y):

   chosen_numbers = lc

   new_numbers = [num for num in range(100) if num not in chosen_numbers]

   if len(new_numbers) > 0:
       return choice(new_numbers)
   else:
       return 0
```

Then, to test this is game_test.py against the other players, I had to make some modifications. First I created variables to track the number of times each player wins.

```python
# Counters for who won
player1 = 0 #caitlyn_jones.py
player2 = 0 #mean.py
player3 = 0 #histo.py
player4 = 0 #prev.py
random_players = 0
no_winner = 0
```

Then, I needed to modify the function so that players 1-4 were using different strategies and players 5-49 were all using random_choice.py.

```python
# Simulate a large number of trials
for i in range(trials):
    cc = [0 for i in range(n)]

    #Player 1 uses my strategy
    cc[0] = ra(caitlyn_jones.play(n, hist, lw, lc, 0))

    #Player 2 uses mean.py
    cc[1] = ra(mean.play(n, hist, lw, lc, 1))

    #Player 3 uses histo.py
    cc[2] = ra(histo.play(n, hist, lw, lc, 2))

    #Player 4 prev.py
    cc[3] = ra(prev.play(n, hist, lw, lc, 3))

    #Last n-4 players use random_choice.py
    for i in range(4,n):
        cc[i] = ra(random_choice.play(n,hist,lw,lc,i))
```

Then, I had to update the scores for each player after each round.

```
    # See who won
    if lw == 100:
        no_winner += 1
    elif cc[0] == lw:
        player1 += 1
    elif cc[1] == lw:
        player2 += 1
    elif cc[2] == lw:
        player3 += 1
    elif cc[3] == lw:
        player4 += 1
    else:
        random_players += 1
```

Lastly, I had to print out the total winning percentages of players 1-4 and the 45 random_choice.py players.

```
# Print the probabilities of winning
f = 100.0/trials
print("Wins for caitlyn_jones strategy :%6.2f%%" % (player1 * f))
print("Wins for mean stratgey :%6.2f%%" % (player2 * f))
print("Wins for histo strategy :%6.2f%%" % (player3 * f))
print("Wins for prev strategy :%6.2f%%" % (player4 * f))
print("Wins for random_choice :%6.2f%% (%.2f%% per player)" % (random_players* f, random_players * f/(n-1)))
print("No winner              :%6.2f%%" % (no_winner * f))
```

```
Wins for caitlyn_jones strategy :  9.51%
Wins for mean stratgey : 15.61%
Wins for histo strategy :  8.35%
Wins for prev strategy :  3.73%
Wins for random_choice : 62.75% (1.31% per player)
No winner              :  0.06%
```

My strategy won 9.51% of the time, which I am happy with as this was the second highest individual one, disregarding the collective score of the random players.