Assignment 8

MATH 381 A - Winter 2022

Yixuan Liu

March 13, 2022

Introduction

In this assignment we will simulate a two-player dice game: Two players alternately roll a die. They can choose to either add the roll to their current score or subtract it from the other player's score. The first player to reach the target score wins.
To win this game, one can have the following strategy:

- always adding, unless their opponent is within n points from the target score and the player themselves will not win the game with that roll, then subtract from the opponent's score

Both player can choose their own value of n. Set the target score to be 15. We will consider strategies with values of n to be $\{0, 1, 2, ..., 6\}$ since the biggest roll a player could add to their score is 6 in one turn, for $n > 6$, the player won't be to worried about losing. We will investigate what value of n a player uses could lead to the highest probability for winning. Assume that if a player choose a strategy, they will not change it whether they are the starter or not. So we will measure the probability of a player to win no matter who starts by averaging the probability with alternate starter.

1

For example, suppose player 1 uses the strategy with $n = 0$, which indicates that player 1 will always adding no matter what score the player 2 gets, and player 2 uses the strategy with $n = 6$. Run 10 simulations with 10000 games in each simulation (see appendix for the Python code to generate the simulation). Plot the probability of player 1 winning in every 100 games for the 10 simulations, we get the following graph

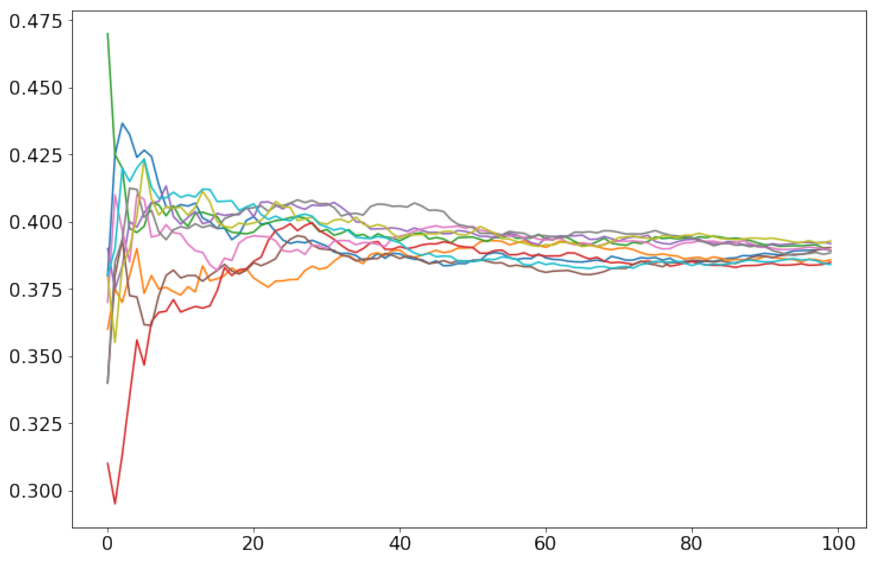

Figure 1: Probability for player 1 to win the game in 10 simulations

We can see that the probabiliies converge at about 0.385. Player 1 is having lower winning probabiliy than player 2, so when player 1 uses $n = 0$ while palyer 2 uses $n = 6$, it is not a good strategy.

2

By Central Limit Theorem, with the same strategy in previous part, we average the probability for player 1 to win in 10000 games and repeat this 10000 times. We plot the averages as a histogram:

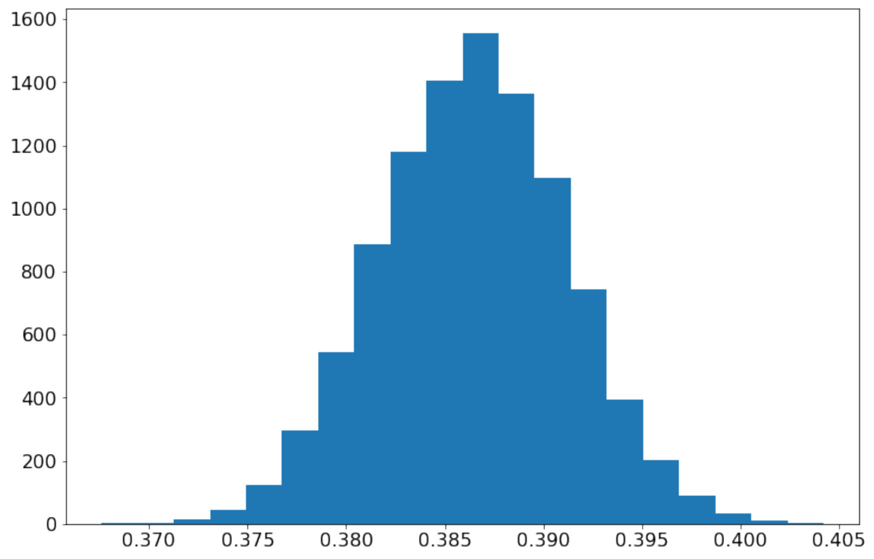

Figure 2: Probability for player 1 to win the game in a 10000 $\times$ 10000 simulations

We can see that the distribution is approximately normally distributed around the mean of the distribution. By Central Limit Theorem, we can calculate a confidence interval for the true probability.

3

To get a more precise confidence interval from previou part, we can use longer runs. So we do another 10 runs of 100000 simulated games. By the weak version of the Central Limit Theorem, the means should be symmetrically distributed with respect to the true probability, so the probability that all 10 of our estimates are on one side of the true probability as

$$\frac{2}{2^{10}} \approx 0.001953125$$

we can conclude that with $\approx$ 99.8% confidence that the true probability lies within the interval $[0.38589, 0.38771]$.

4

Now we can compare different strategy pairs for both player choosing $n \in \{0, 1, ..., 6\}$. Repeat the same procedure in section 3 but different strategy pairs, we can create a table of confidence intervals for the probability that player 1 wins:

player 1	player 2							
	0	1	2	3	4	5	6	7
0	(0.49577,0.50282)	(0.48415,0.48929)	(0.46775,0.47299)	(0.44799,0.4532)	(0.42516,0.42865)	(0.40122,0.40658)	(0.38474,0.38884)	
1	(0.5104,0.51523)	(0.49779,0.50175)	(0.48231,0.48525)	(0.46106,0.46594)	(0.43553,0.44029)	(0.40921,0.41415)	(0.38997,0.3933)	
2	(0.52906,0.53365)	(0.51485,0.51804)	(0.49686,0.50132)	(0.47705,0.48024)	(0.44955,0.45498)	(0.42004,0.42565)	(0.39628,0.40142)	
3	(0.54821,0.55027)	(0.53537,0.53963)	(0.51943,0.52255)	(0.49837,0.50304)	(0.47039,0.47424)	(0.43799,0.44178)	(0.40903,0.4142)	
4	(0.5709,0.5751)	(0.55908,0.56372)	(0.54568,0.54908)	(0.52426,0.52987)	(0.49769,0.50188)	(0.46327,0.46706)	(0.42737,0.43247)	
5	(0.59479,0.59727)	(0.58687,0.58864)	(0.57565,0.58193)	(0.55747,0.56295)	(0.53377,0.53736)	(0.49607,0.50258)	(0.46073,0.46557)	
6	(0.61035,0.61675)	(0.6056,0.61117)	(0.59829,0.60167)	(0.58522,0.58931)	(0.56388,0.56952)	(0.53426,0.53778)	(0.49724,0.50242)	
7	(0.60362,0.60843)	(0.59753,0.60347)	(0.59194,0.59759)	(0.58072,0.58821)	(0.56314,0.5664)	(0.52632,0.53176)	(0.48427,0.48802)	/
8	(0.59544,0.59902)	(0.5921,0.59643)	(0.58837,0.59157)	(0.57704,0.58064)	(0.55809,0.56285)	(0.51942,0.52419)	(0.46805,0.47356)	(0.48251,0.48801)
9	(0.58566,0.59048)	(0.58469,0.58882)						

From the table we can see that as long as player 1 choose higher n value than player 2, they will have higher probability to win the game than player 2. We can also see that when player 1 choose $n = 6$, the probability will always be greater than or approximately same as plaer 2. So for a player to win the game, the best strategy is to choose $n = 6$ no matter what strategy the other player choose, and the probability to win the game will not be lower than their opponent. What about strategies with n>6 since one can subtract from their oppenent's score if the oppenent could win within 2 turns? We can see that the probability intervals are symmetrical along when player 1 and player 2 have same value of n, and when player 1 and player 2 are having same value of n, the probability is always about 0.5, so we can get a reduced table. We can see that when n>6, as player 1 has bigger n, their winning probability decreases, so for player 1, the strategy with $n = 6$ is still the best strategy.

5

Now consider another strategy with the following rule:

- always adding, unless their opponent's score subtract their own score is within n score and the player themselves will not win the game with that roll, then subtract from the opponent's score

With this rule, we assume that the players will always use $n > 2$ since when $n \leq 2$ some cases will take forever for the game to run by simulation. Do simular simulation from section 4, we get the following table of confidence intervals for the probability that the player 1 wins:

player 1	player 2						
	3	4	5	6	7	8	9
3	(0.49793,0.50244)	(0.51397,0.51776)	(0.53946,0.54418)	(0.56916,0.57266)	(0.5941,0.5982)	(0.61246,0.61649)	(0.62241,0.62851)
4	(0.48296,0.49063)	(0.49843,0.50238)	(0.52169,0.52778)	(0.54943,0.55375)	(0.57474,0.58058)	(0.59049,0.59485)	(0.59784,0.60367)
5	(0.45535,0.45969)	(0.47179,0.47697)	(0.49727,0.50074)	(0.52439,0.53035)	(0.54702,0.55147)	(0.56248,0.56626)	(0.57038,0.57492)
6	(0.42559,0.42996)	(0.44526,0.45113)	(0.47101,0.47615)	(0.49911,0.50298)	(0.51875,0.52548)	(0.53503,0.5406)	(0.543,0.54765)

We can see that the smaller n the player 1 chooses, they will win with higher probability. In addition, is player 1's n is smaller than player 2/s n, player 1 will have higher winning probability than player 2. Even though we did not include the strategies with $n \leq 2$, we can infer that if player 1 uses strategy wih $n \leq 2$ will lead to higher winning probability. Howeverm when player 1 has $n \leq 2$, we have the following table of probability for player 1 to win:

player 1	player 2						
	3	4	5	6	7	8	9
0			(0.4547,0.4739)	(0.4957,0.5093)	(0.5577,0.5685)	(0.5975,0.6085)	(0.626,0.642)
1		(0.4779,0.4972)	(0.5037,0.5173)	(0.5477,0.5676)	(0.5885,0.6077)	(0.6221,0.6384)	(0.6371,0.6502)
2	(0.4827,0.5024)	(0.4916,0.5174)	(0.5342,0.5499)	(0.5681,0.5881)	(0.6007,0.6146)	(0.6204,0.6322)	(0.631,0.6504)

Combined the above two tables, we can see that there is not a pattern of the probability for player 1 to win. The general strategy for player 1 to win is to use $n < 7$ while player 2 has $n \geq 7$, and when player 2 has $n < 7$, player 1 will not have notable higher proabability to win compare to player 2.

1 Appendix A - Python code for section 1 to section 4

Python code to generate the simulaton of the dice game:

```
import numpy as np
import random
import matplotlib.pyplot as plt

target = 20

def simulation(strategy1, strategy2, games):
  turns = 0
  win1 = 0
  win2 = 0
  win_convergence = np.zeros(int(games/100))
  for i in range(int(games/2)):
    # when player 1 start
    score1 = 0
    score2 = 0
    while score1 < target and score2 < target:
      roll1 = random.randint(1,6)
      roll2 = random.randint(1,6)

      if (score1 + roll1) >= target:
        score1 += roll1
      elif (target - score2) <= strategy1:
        score2 -= roll1
      else:
        score1 += roll1

      # if player 1 hits the target score, end the game
      if score1 >= target:
        break

      if (score2 + roll2) >= target:
        score2 += roll2
      elif (target - score1) <= strategy2:
        score1 -= roll2
      else:
        score2 += roll2

      turns += 1
    if score1 >= target:
      win1 += 1
    if score2 >= target:
      win2 += 1
    # when player 2 start
    score1 = 0
    score2 = 0
    while score1 < target and score2 < target:
      roll1 = random.randint(1,6)
      roll2 = random.randint(1,6)

      if (score2 + roll2) >= target:
        score2 += roll2
      elif (target - score1) <= strategy2:
        score1 -= roll2
      else:
        score2 += roll2

      # if player 2 hits the target score, end the game
      if score2 >= target:
        break

      if (score1 + roll1) >= target:
        score1 += roll1
      elif (target - score2) <= strategy1:
        score2 -= roll1
      else:
        score1 += roll1

      turns += 1
    if score1 >= target:
      win1 += 1
    if score2 >= target:
      win2 += 1

    # record the probability for player 1 to win every 100 games
    if (i+1)*2/100 == int((i+1)*2/100):
      win_convergence[int((i+1)*2/100)-1] = win1/((i+1)*2)
  prob_win1 = win1/games
  prob_win2 = win2/games
  mean_turns = turns/games
  #print(prob_win1, prob_win2, mean_turns)
  return win_convergence, prob_win1

#plot convergence
fig, ax1 = plt.subplots(figsize=(12,8))
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
for i in range(10):
  [x,y] = simulation(0, 6, 10000)
  #print(win_player1)
  plt.plot(range(100),x)
```

```
#plot histogram
win_hist = np.zeros(10000)
for i in range(10000):
  [x,y] = simulation(0, 6, 10000)
  win_hist[i] = y

fig, ax1 = plt.subplots(figsize=(12,8))
plt.xticks(fontsize=16)
plt.yticks(fontsize=16)
plt.hist(win_hist, bins = 20)
plt.show

mean = np.mean(win_hist)
std = np.std(win_hist)
print(mean, std)
print(np.min(win_hist), np.max(win_hist))

# get intervals
win_int = np.zeros(10)
for i in range(10):
  [x,y] = simulation(0, 6, 100000)
  win_int[i] = y

print(np.min(win_int), np.max(win_int))

# get intervals for player 1 to win in different strategy pairs
for i in range(7):
  for j in range(7):
    win = np.zeros(10)
    for k in range(10):
      [x,y] = simulation(i, j, 100000)
      win[k] = y
    print(i, j, "(", np.min(win), ",", np.max(win), ")", sep = "")
```

2 Appendix B - Python code for section 5

Python code to generate the simulaton of the dice game:

```
def simulation(strategy1, strategy2, games):
  turns = 0
  win1 = 0
  win2 = 0
  win_convergence = np.zeros(int(games/100))
  for i in range(int(games/2)):
    # when player 1 start
    score1 = 0
    score2 = 0
```

```
while score1 < target and score2 < target:
  roll1 = random.randint(1,6)
  roll2 = random.randint(1,6)

  if (score1 + roll1) >= target:
    score1 += roll1
  elif (score2 - score1) >= strategy1:
    score2 -= roll1
  else:
    score1 += roll1

  # if player 1 hits the target score, end the game
  if score1 >= target:
    break

  if (score2 + roll2) >= target:
    score2 += roll2
  elif (score1 - score2) >= strategy2:
    score1 -= roll2
  else:
    score2 += roll2

  turns += 1
if score1 >= target:
  win1 += 1
if score2 >= target:
  win2 += 1
# when player 2 start
score1 = 0
score2 = 0
while score1 < target and score2 < target:
  roll1 = random.randint(1,6)
  roll2 = random.randint(1,6)

  if (score2 + roll2) >= target:
    score2 += roll2
  elif (score1 - score2) >= strategy2:
    score1 -= roll2
  else:
    score2 += roll2

  # if player 2 hits the target score, end the game
  if score2 >= target:
    break

  if (score1 + roll1) >= target:
    score1 += roll1
```

```
      elif (score2 - score1) >= strategy1:
        score2 -= roll1
      else:
        score1 += roll1

      turns += 1
    if score1 >= target:
      win1 += 1
    if score2 >= target:
      win2 += 1

    # record the probability for player 1 to win every 100 games
    if (i+1)*2/100 == int((i+1)*2/100):
      win_convergence[int((i+1)*2/100)-1] = win1/((i+1)*2)
  prob_win1 = win1/games
  prob_win2 = win2/games
  mean_turns = turns/games
  #print(prob_win1, prob_win2, mean_turns)
  return win_convergence, prob_win1

for k in range(10):
  [x,y] = simulation(7, 0, 1000000)
  win[k] = y
print("[",np.min(win),",",np.max(win),"]",sep = "")
```