Phan Binh Nguyen (21365839), Emeka David Odoemelam (20334547)

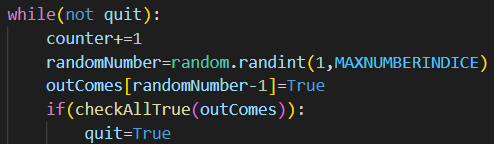
02/12/2022

ST2004  
applied probability I group assignment

**Probability Assignment**

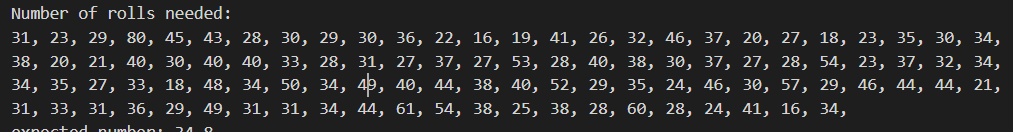
Question 1.

Task: A 12-sided die is rolled continuously until all the possible outcomes have occurred at least  
once.

**A,** *Estimate the expected number of dice rolls needed using a simulation study.*

**Algorithm:** We simulate the dice by generating random numbers between 1 and 12. “outComes” is a Boolean list with size 12; at first, it is filled up with 12 false. In each roll, we generate a random number between 1 and 12 and then change the “randomNumber-1”-th item in the list to true. The while loop quits if all the values in the last are "true" i.e., all the possible outcomes have occurred.

Using simulations found in q1.py running 100 of simulations, it returns an expected number of 34.8, with number of rolls shown in this picture.



Chart, line chart

Description automatically generatedRunning the same simulation 10000 times would result a more precise expected number, which is 37.518. The average of the rolls needed to hit all the possible outcomes, approaches the real expected number by each simulation.0020

**B,** *Compute the expected number of dice rolls needed analytically.*

In this case we are calculating the chance of rolling a side of the dice we have yet to roll before. Since the rolls are independent and each side has the same chance of being rolled, we use a geometric distribution. (Values a shortened to 3 decimal places)

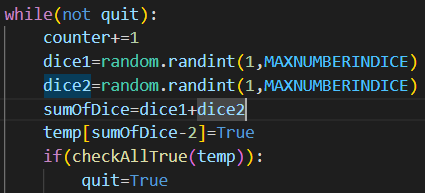
1. On the first roll we have a probability of 1 to land on a side of the dice we have yet to see. **Expected rolls: = 1**
2. Following the first roll we have 11 sides remaining. For every successive roll we have an chance of seeing a new side. **Expected rolls: = 1.09**
3. Following the appearance of 2 different values we have 10 more left. For every successive roll we have an chance of seeing a new side. **Expected rolls: = 1.2**
4. Following the appearance of 3 different values we have 9 more left. For every successive roll we have an chance of seeing a new side. **Expected rolls: = 1.333**
5. 8 values left with an chance of seeing a new side. **Expected rolls: = 1.5**
6. 7 values left with an chance of seeing a new side. **Expected rolls: = 1.714**
7. 6 values left with an chance of seeing a new side. **Expected rolls: = 2**
8. 5 values left with an chance of seeing a new side. **Expected rolls: = 2.4**
9. 4 values left with an chance of seeing a new side. **Expected rolls: = 3**
10. 3 values left with an chance of seeing a new side. **Expected rolls: = 4**
11. 2 values left with an chance of seeing a new side. **Expected rolls: = 6**
12. Single value left with an chance of seeing a new side. **Expected rolls: = 12**

Total expected rolls equals to the sum of the above cases.

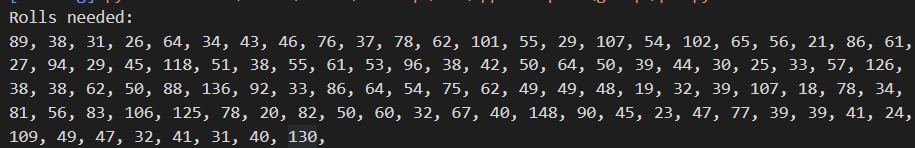
**Total expected rolls:** 1 + 1.09 + 1.2 + 1.333 + 1.5 + 1.714 + 2 + 2.4 + 3 + 4 + 6 + 12 **= 37.237**

The simulations value for number of expected rolls: 37.518. Calculated value: **37.237**

These two values are close enough to assume that the simulation model works well.

****Task: “*A pair of 6-sided fair dice are rolled continuously until all the possible outcomes (i.e. all  
possible sums of two dice, 2, 3, . . . , 12) have occurred at least once. Estimate the expected  
number of dice rolls needed using a simulation study. “*

**Algorithm:** We simulate the two dice (dice1 and dice2) with two random number generators that generate a random number between 1 and 6. Temp is a Boolean list that has a size of 11. (All the possible sums, 2-12). In each round, we sum up the two rolls and change the “sum-2”-th element to "true" since the indexing of the list starts at 0. We exit the loop if all the possible sums have occurred.

Using simulations found in q1B.py running 100 of simulations, it returns an expected number of 59.09, with number of rolls shown in this picture.

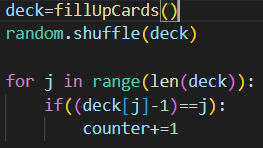
Chart, line chart

Description automatically generatedRunning the same simulation 10000 times would result a more precise expected number, which is 61.17. The average of the rolls needed to hit all the possible sums, approaches the real expected number by each simulation.

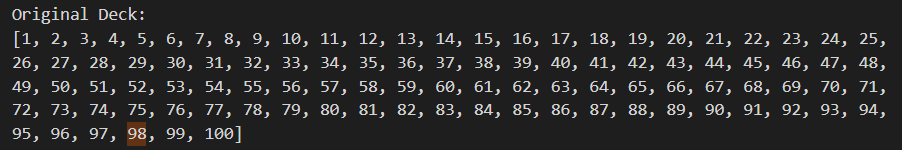
Question 2.

Task “A deck of 100 cards - numbered 1, 2, . . . , 100 - is shuffl­ed and then turned over one card at a  
time. We say that a “hit” occurs whenever card i is the ith card to be turned over, i = 1, . . . , 100.  
Simulate 10 000 repetitions of the game to estimate the expectation and variance of the total  
number of hits”

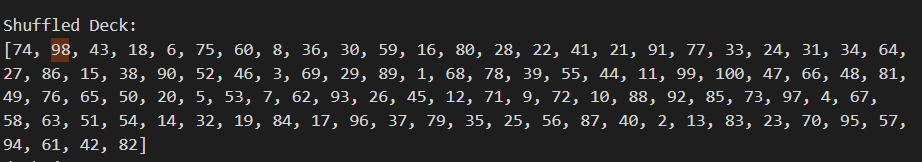
**Algorithm:**

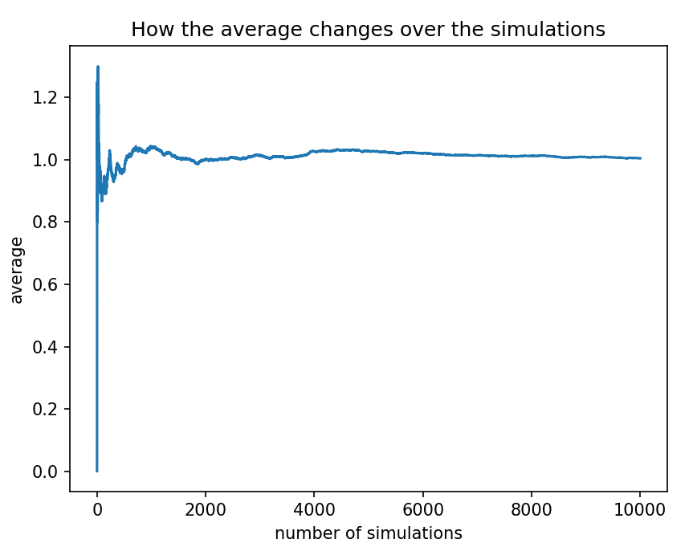
In each round, we create a new list (deck) with size 100, and we fill it up with numbers ranging from 1 to 100 that represent the 100 cards.

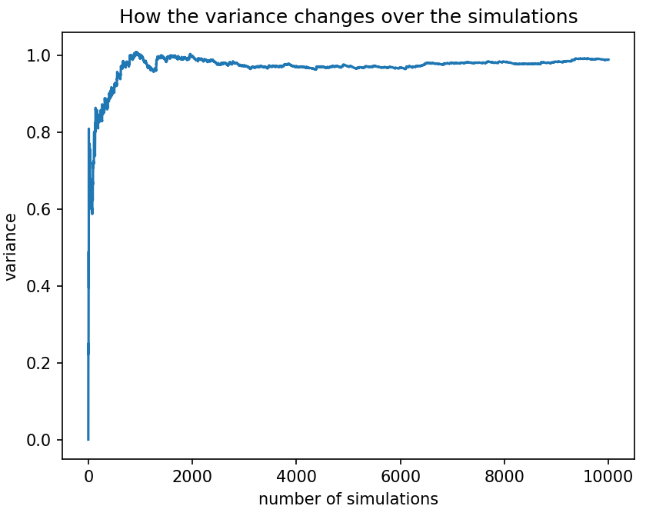
We shuffle the list, and then iterate through the list. Each time we compare the counter (j) to the „j-1”th element in the list since the counter starts at 0 while the numbers stored in the list start from 1.Unshuffled deck:



Same deck after being shuffled:

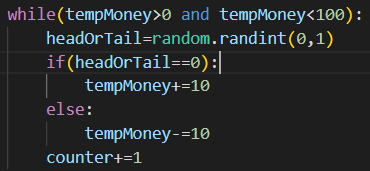


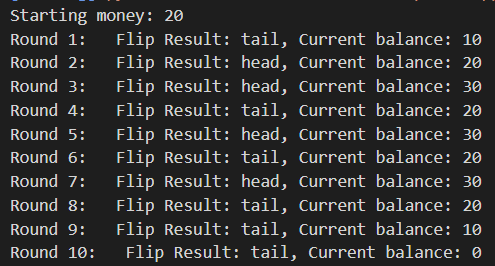
 In this case, we can observe four hits, namely cards 8, 39, 49, and 95, which brings the average to 4. However, if we do the simulation, found in q2.py 10 000 times, we can observe a number closer to the real expected number. The expected number of hits over 10 000 simulations is approx.**: 1.0052**.

Question 3.

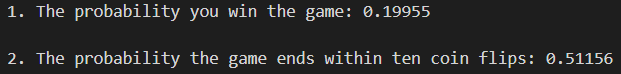
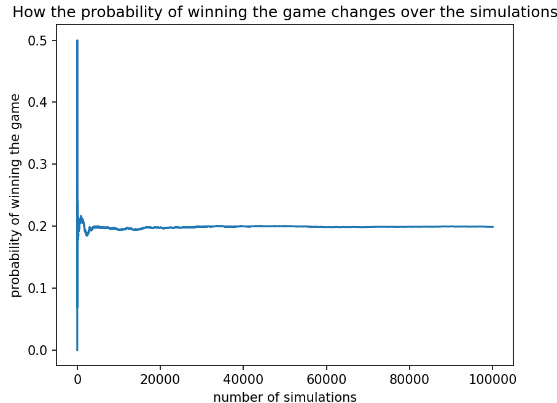
Task*: Consider the following game: you begin with $20. You flip a coin, winning $10 if the coin lands  
heads and losing $10 if the coin lands tails. The game is played continuously until you either go  
broke or have $100 (i.e. a net profit of $80). Estimate using simulation studies  
1. The probability you win the game.  
2. The probability the game ends within ten coin flips.*

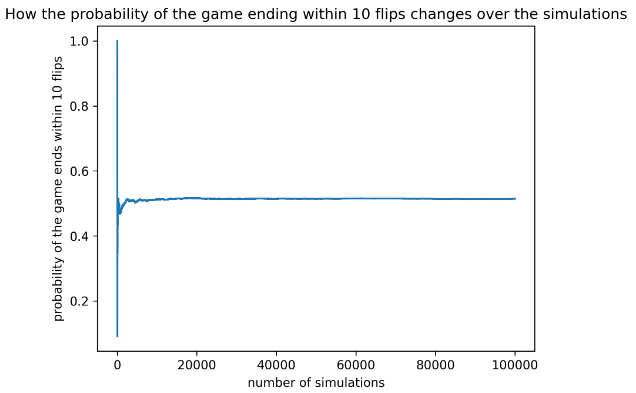
**Algorithm:**

In each simulation, we assign tempMoney to 20, which is the starting balance. We simulate the coin flip by generating random numbers between 0 and 1. If the random number is 0, we simulate head, i.e., the player gets 10$; if the random number is one, we simulate tail, i.e., the player loses 10$. The counter counts how many rounds it needs until the player wins (balance is equal to 100) or loses (balance is equal to 0). If either a win or loss occurs, the loop ends.

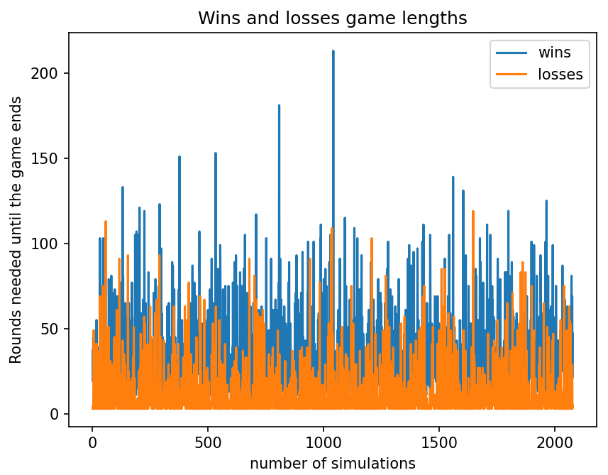
In this case we simulated that the player loses within 10 coin flips.

To get the probability of winning the game and the probability of the game ends within 10 coin flips using the simulation found in q3.py, we can simulate the game 10000 times, we get:

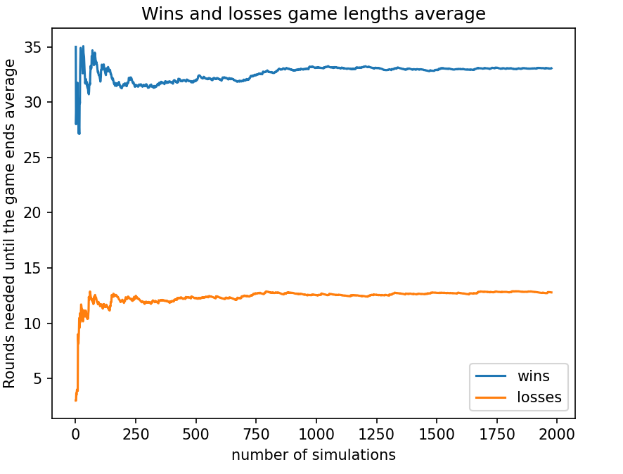




In the plots above, we can observe that with more simulation, we can get a more precise approximation as the line flattens.

Since the starting balance is closer to the losing end than the winning end, we can observe that, on average, it takes more rounds for a winning game than for a losing game. In the picture to the left, we can observe that even though some losing game’s length could spike up, most losing games end under 50 rounds.

If we take the average of the rounds it takes to win or lose, we can plot it like this

Where the number of rounds it takes to win is approx. 32.6, while the average rounds it takes to lose is only around 12.4. We can assume that most of the games that ended within ten rounds were losses. Therefore, the more game we lose, the number of games that end within ten rounds gets larger.

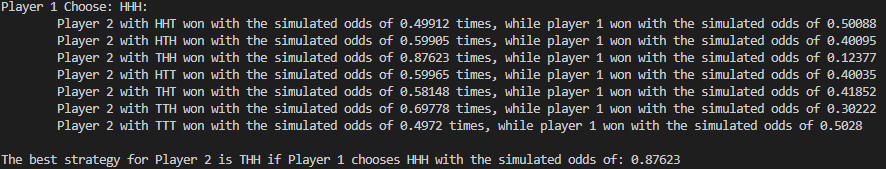
Question 4.

Using simulations found in appProbq4.java running 100000 of simulations for each strategy option for player 2 (P2) given players 1 (P1) choice and looking at the number of wins we can get an idea of the best strategy for each choice of player 1.

H = heads while T = tails.

**Case 1 player 1 selects HHH as their pattern.**

The simulation returns the following results:



This suggests the optimal strategy in case 1 is for player 2 to select THH as their pattern.

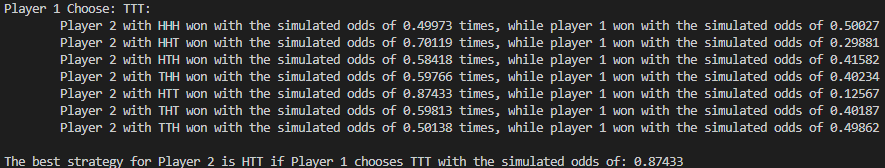
Given these two selections of P1: HHH and P2: THH we can see that the only way P1 can win is if their pattern emerges immediately in the first 3 flips. This is because if there is a single T or tails flipped it resets P1 to needing 3 H in a row to win but P2 only needs 2 in a row meaning P1 cannot win.

Calculations:

Meaning with the optimal strategy of THH P2 gives the player a chance of winning of . This is in line with the result of the simulation.

**Case 8 player 1 selects TTT as their pattern.**

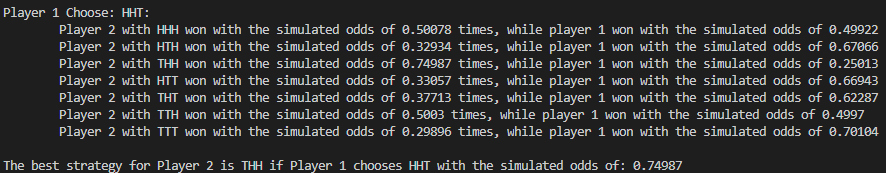
The simulation returns the following results:



This case is effectively identical to case 1 with Ts swapped for Hs meaning the optimal strategy is HTT and the chance of P2 winning is the identical . This is also lines up with the simulation.

**Case 2 player 1 selects HHT as their pattern.**

The simulation returns the following results:



This suggests the optimal strategy in case 2 is for player 2 to select THH as their pattern.

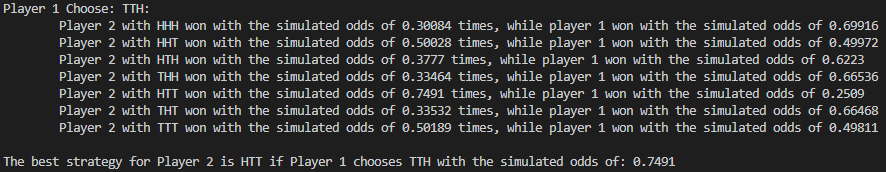
Similarly, to case 1 we can once again observe the limitations on P1 winning the game. P1 needs the first two flips to be H. If any of the first two flips is T it resets P1 and puts P2 in an advantage and since at this point P2 needs two consecutive Hs the same that P1 needs to get the first two elements of their pattern they can no longer win. P1’s chance of winning is the probability of the first two flips being H.

Calculations:

Meaning with the optimal strategy of THH P2 gives the player a chance of winning of . This is in line with the result of the simulation.

**Case 7 player 1 selects TTH as their pattern.**

The simulation returns the following results:

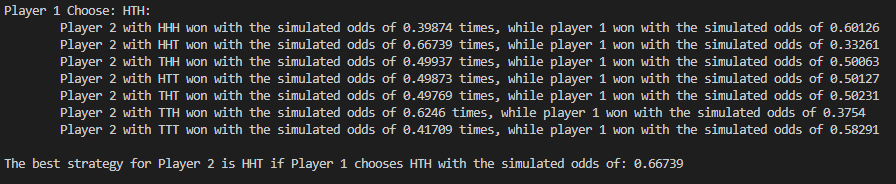


This is essentially the same as case 2 with the patterns inverted meaning the calculations and the result will be identical while the simulation should be similar, this is the case.

The optimal strategy HTT gives P2 the chance of winning of , inline with the simulation results.

**Case 3 player 1 selects HTH as their pattern.**

The simulation returns the following results:

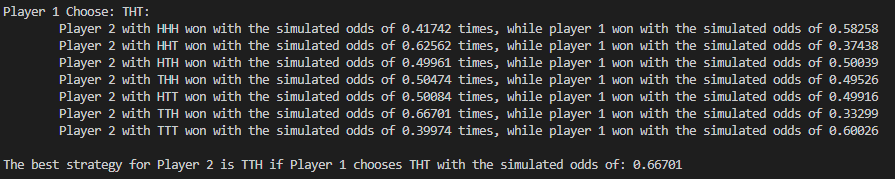


This suggests the optimal strategy in case 2 is for player 2 to select HHT as their pattern.

In this case we end up with a situation where both players need the first flip from the beginning of a new sequence to be H. Following this we branch into the two directions if the second flip is also H P2 wins since at some point a T will come while H do not change the pattern the chance for this is while in the other case P1 still needs the next flip to be H to win because if its T then both players are reset, and we essentially return to the starting state to begin anew. Since the third flip is also determined by the probability we get the following

Returning to start is irrelevant to us. Since P2 is twice as likely to win their probability of winning using optimal strategy HHT is . This is in line with simulations.

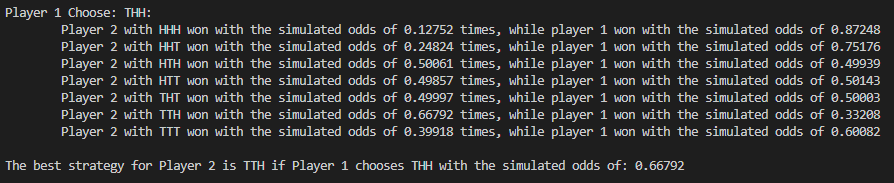
**Case 6 player 1 selects THT as their pattern.**



Once again, we have an invers this time of case 3 leading to the same result and similar simulation results. The optimal strategy is TTH for P2 and their odds of winning are .

**Case 4 player 1 selects THH as their pattern.**

The simulation returns the following results:

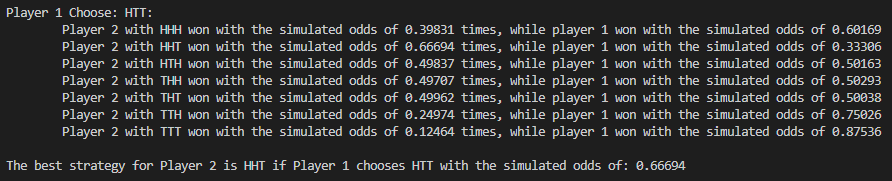


This suggests the optimal strategy in case 2 is for player 2 to select TTH as their pattern.

This case is essentially identical to case 3 once again both players need the first flip to be T. If it is H it is meaningless as it does not benefit any player. Once the first T arises P2 needs the next one to be T as well after that no matter how many more Ts arise they will win with the next H. Meanwhile P1 still needs the third flip to go in their favor otherwise we return to start with both players “wiped” giving us the same probabilities of:

Once again P2’s chance of winning with the optimal strategy of TTH is in line with simulations.

**Case 5 player 1 selects THH as their pattern.**



Once again, an inverse this time of case 4. Optimal strategy id HHT with the probability of winning being as the simulation predicts.

**Flowcharts for Q4** (Each node splits into two with a chance in going eighter way.)

Start

P1 wins

P2 wins

Case 1 and Case 8

Start

P1 wins

P2 wins

Case 2 and Case 7

Start

P2 wins

P1 wins

Case 3 and Case 6

As well as

Case 4 and Case 5

**Pseudo Code for Q4 simulation**

for(Every choice of P1){

int [][] wins

for(every choice of P2){

char[3] sequence

if(not the same choice){

for(100000 times){

while(we get a winner){

shift sequence left to prepare for new flip result

flip coin and store result in sequence[2]

compare sequence to P1 and P2 selection to se if there is a winner

if(there is a winner){

increase appropriate win counter in wins

}

}

}

}

print out results for all 7 of P2’s options and their win rates against P1 stored in wins

highlight best strategy

}

}