Sam Hopkins
CS 470
Lab 1 - Probability

1. St. Petersburg Paradox:
    a. 100
        i. Average: $8.10
        ii. Max: $128
    b. 10000
        i. Average: $20.125
        ii. Max: $32768
    c. 1000000
        i. Average: $22.75
        ii. Max: $2097152

    Assuming that I could play as much as I want, if I were playing upward of 1 million times, I can safely pay up to $20, and safely assume that at worst case I will gain $2.75 (according to the probability).

2. Monty Hall
    a. 1000 Games with not switching strategy: 333 cars won out of 1000 games (%33)
    b. 1000 games with switching: 670 cars won out of 1000 games (%67)
    c. Un-estimated percentage:

Sam Hopkins
CS 470
Lab 1 - Probability

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

we pick A and stay

Given we pick A

Car is in B

$$P(B) = P(B|A)P(A) + P(B|B)P(B) + P(B|C)P(C)$$

Monty opens B

$$P(B) = \frac{1}{2} \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3}$$

Given we pick A

$$P(B) = \frac{1}{6} + \frac{1}{3} = \frac{1}{2}$$

$$\frac{P(B|A)P(A)}{\frac{1}{2}} = \frac{\cancel{\frac{1}{2}} \cdot \frac{1}{3}}{\cancel{\frac{1}{2}}} = \boxed{\frac{1}{3}}$$ if we stay

$$P(C|B) = \frac{P(B|C)P(C)}{P(B) = \frac{1}{2}}$$

Prob car is in C if we pick A

$$P(C|B) = \frac{1 \cdot \frac{1}{3}}{\frac{1}{2}} = \boxed{\frac{2}{3}}$$ if we change

3. RISK
    a. Player outcomes with na and nd:

Sam Hopkins
CS 470
Lab 1 - Probability

i. na = 3, nd = 2
ii. (1, 1) = 336446 /1000000 : 34 % Both Loose 1
iii. (2, 0) = 292113 /1000000 : 29 % Attacker looses 2
iv. (0, 2) = 371441 /1000000 : 37 % Defender looses 2

v. na = 2, nd = 2
vi. (1, 1) = 324387 /1000000 : 32 % Both Loose 1
vii. (2, 0) = 448497 /1000000 : 45 % Attacker looses 2
viii. (0, 2) = 227116 /1000000 : 23 % Defender looses 2

ix. na = 1, nd = 2
x. (1, 0) = 746223 /1000000 : 75 % Attacker looses 1
xi. (0, 1) = 253777 /1000000 : 25 % Defender looses 1

xii. na = 3, nd = 1
xiii. (1, 0) = 340130 /1000000 : 34 % Attacker looses 1
xiv. (0, 1) = 659870 /1000000 : 66 % Defender looses 1

xv. na = 2, nd = 1
xvi. (1, 0) = 420679 /1000000 : 42 % Attacker looses 1
xvii. (0, 1) = 579321 /1000000 : 58 % Defender looses 1

xviii. na = 1, nd = 1
xix. (1, 0) = 584647 /1000000 : 58 % Attacker looses 1
xx. (0, 1) = 415353 /1000000 : 42 % Defender looses 1

According to the data, there is no reason that an attacker or defender should not use all the dice available. The times when the probability of losing an army or two armies is minimized is when the player uses as many dice as possible.

I used 1 million samples in order to get a more exact estimate, as I would like to know the overall probability as accurate as possible for each situation.

b. Attacker win percentage with A=n armies and D=5 armies (out of 1 million trials for each). I used 1 million samples in order to estimate as close as possible, and 1 million will have a good estimate for each proabability.

i. A = 2
ii. 9922 wins, .99 %

iii. A = 3
iv. 79148 wins, 7.9148 %

v. A = 4
vi. 206177 wins, 20.62 %

Sam Hopkins
CS 470
Lab 1 - Probability

vii. A = 5
viii. 358007 wins, 35.80 %

ix. A = 6
x. 506663 wins, 50.67 %

xi. A = 7
xii. 638118 wins, 63.81 %

xiii. A = 8
xiv. 736487 wins, 73.65 %

xv. A = 9
xvi. 818413 wins, 81.84 %

xvii. A = 10
xviii. 873272 wins, 87.33 %

xix. A = 11
xx. 916080 wins, 91.61 %

xxi. A = 12
xxii. 943081 wins, 94.31 %

xxiii. A = 13
xxiv. 964017 wins, 96.40 %

xxv. A = 14
xxvi. 975779 wins, 97.58 %

xxvii. A = 15
xxviii. 985092 wins, 98.51 %

xxix. A = 16
xxx. 990008 wins, 99.00 %

xxxi. A = 17
xxxii. 994008 wins, 99.40 %

xxxiii. A = 18
xxxiv. 996171 wins, 99.62 %

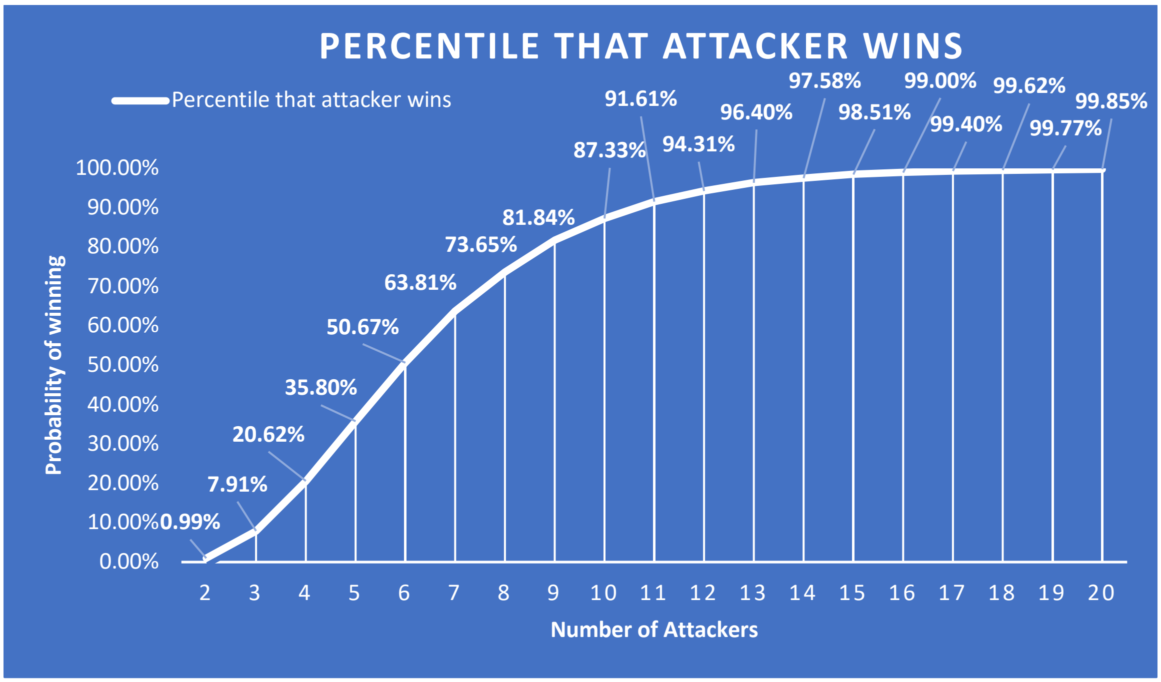
Sam Hopkins
CS 470
Lab 1 - Probability

xxxv. A = 19

xxxvi. 997657 wins, 99.77 %

xxxvii. A = 20

xxxviii. 998528 wins, 99.85 %



According to the data, if I wanted a guaranteed win percentage of 50%, I would need to attack a defender (with 5 defending armies) with 6 armies. This yields an average percentage of 50.64% win rate, and after multiple trials it has never dropped below 50%. For 80%, I would need 9 armies.

I also used 1 million samples because I wanted to find exact probability, and the higher the sample size the closer I would get to the exact probability.

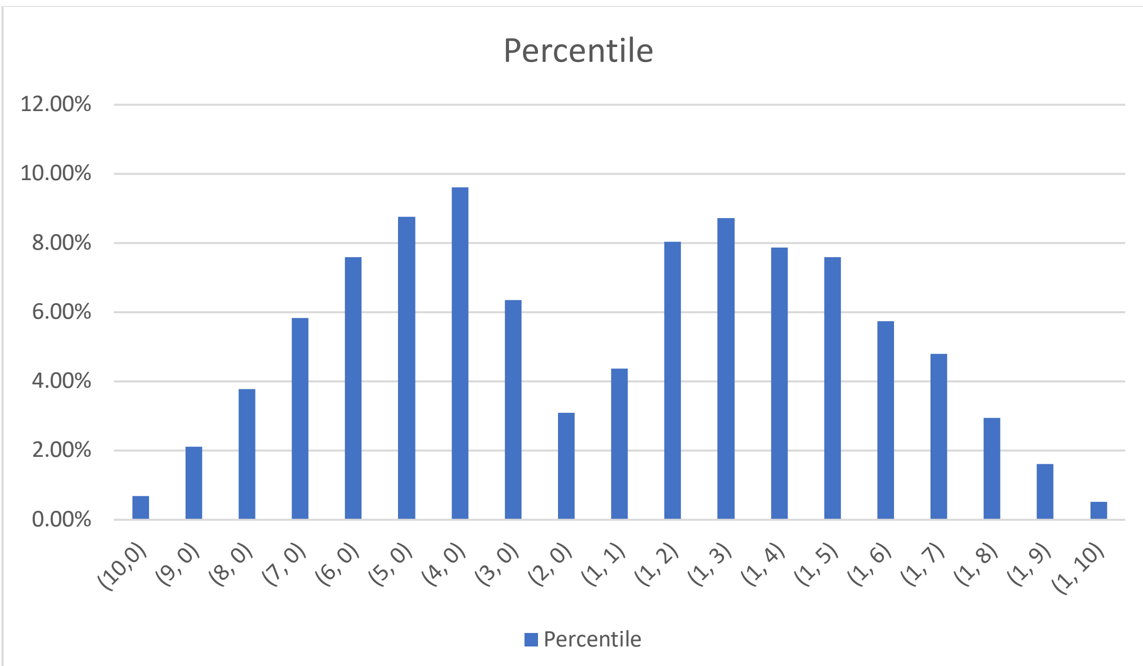
# of Attackers	# of Defenders	Percentile
10	0	0.68%
9	0	2.11%
8	0	3.77%
7	0	5.83%
6	0	7.59%
5	0	8.76%
4	0	9.61%
3	0	6.35%
2	0	3.10%

Sam Hopkins
CS 470
Lab 1 - Probability

1	1	4.37%
1	2	8.04%
1	3	8.72%
1	4	7.87%
1	5	7.59%
1	6	5.75%
1	7	4.79%
1	8	2.95%
1	9	1.61%
1	10	0.51%

Percentage that Attacker wins: 47.8%
Percentage that Defender wins: 52.2%

Plot:



I used 1 million samples for this section too, so I could get as accurate percentages as I could. I wanted to estimate as close as possible.

Sam Hopkins
CS 470
Lab 1 - Probability
Code:

```
import random
import matplotlib.pyplot as plt

def roleDice():
    return random.randint(1,6)

def risk(attackerArmies, defenderArmies, attackerDice=3, defenderDice=2):
    # print("# of Attackers:", attackerArmies)
    # print("# of Defenders:", defenderArmies)

    while attackerArmies > 1 and defenderArmies > 0:
        attackerRole = []
        defenderRole = []
        for x in range(attackerDice):
            attackerRole.append(roleDice())
        attackerRole.sort()
        attackerRole.reverse()

        for x in range(defenderDice):
            defenderRole.append(roleDice())
        defenderRole.sort()
        defenderRole.reverse()

        for x in range(min(len(defenderRole), len(attackerRole))):
            if defenderRole[x] < attackerRole[x]:
                defenderArmies -= 1
            else:
                attackerArmies -= 1

        attackerDice = min(3, attackerArmies - 1)
        defenderDice = min(2, defenderArmies)

    return (attackerArmies, defenderArmies)

def numArmiesLost(attackerDice=3, defenderDice=2):
        attackerRole = []
        defenderRole = []
        attackerArmiesLost = 0
        defenderArmiesLost = 0
        for x in range(attackerDice):
            attackerRole.append(roleDice())
        attackerRole.sort()
        attackerRole.reverse()

        for x in range(defenderDice):
            defenderRole.append(roleDice())
        defenderRole.sort()
        defenderRole.reverse()

        for x in range(min(len(defenderRole), len(attackerRole))):
            if defenderRole[x] < attackerRole[x]:
                defenderArmiesLost += 1
            else:
                attackerArmiesLost += 1
        return (attackerArmiesLost, defenderArmiesLost)
```

Sam Hopkins
CS 470
Lab 1 - Probability

```
# Monty Hall Solution
def montyHall():
    doors = [0, 0, 1] # 1 is car, 0's are goats
    random.shuffle(doors)
    goatIndexes = []
    for x in range(len(doors)):
        if doors[x] == 0:
            goatIndexes.append(x)

    doorChoice = random.randint(0,2)
    randomGoat = goatIndexes[random.randint(0,1)]

    if(doorChoice == randomGoat):
        randomGoat = (randomGoat + 1) % 2

    if(doors[doorChoice] == 1):     # 1 if I didn't need to switch
        return 1
    return 0                        # 0 if switch is needed



# St Peter's Paradox solution
def stPetersParadox():
    numHeads = 0
    coinFlip = 0          # O = Heads, 1 = Tails
    while coinFlip == 0:
        numHeads += 1
        coinFlip = random.randint(0,1)
    return pow(2, numHeads)


if __name__ == '__main__':
    # for x in range(5, 21):
    # print("A =", x)
    list = []
    d = {}
    numTimes = 1000000
    # for y in range(numTimes):
    #     result = risk(6, 5)
    #     list.append(result)

    # percent = "{:.2f}".format((sum(list)/numTimes) * 100)
    # # print("Results for", numTimes, "runs: Attacker wins ", sum(list))
    # print(sum(list), "wins,", percent, "%")
    # print()


    for x in range(numTimes):
        key = risk(10,10)
        if key not in d:
            d[key] = 1
        else:
            d[key] += 1

    for key in d:
        percent = "{:.2f}".format((d[key] / numTimes) * 100)
        print(key, "=", d[key], f"/{numTimes}  : ", percent, "%")
```