Assignment #6: Simulation of Cee-lo

Xinmin Zhang, Samuel Han

For our simulation, we will be looking into a dice game called "Cee-lo", originated from east Asia. The name comes from the pronunciation of "four-five-six" in Chinese. There is a variety of existing rules for this game. Here we will introduce a slightly modified version.

The game involves 2 players, each having at most 3 rounds to roll 3 six-sided dice, with the aim to reach a biggest combination possible.

Attaining a combination of (4,5,6) from the three dice is the best possible outcome, and thus leads to instant win.

Following that, "Baozi", or three of a kind, is the next best possible combination. When comparing, (6,6,6) is better than (5,5,5), which is better than (4,4,4), and all the way down to (1,1,1) accordingly.

Then, if one rolls two of a kind, the third die which is singled out would be counted as the player's score. This is to say that a combination of (1,1,5) is better than (6,6,2), as the pair isn't considered when comparing the result, but only the single die's face value matters.

Then, on the contrary of (4,5,6), if one rolls a (1,2,3), it leads to instant loss, no matter if the player has one or two more rounds remaining to reroll the dice.

If after three rounds, none of the above combination was achieved, the player's score would simply be zero, but still would be considered to win if his or her opponent rolled a (1,2,3).

For our game, although each player is granted with three rounds to roll their dice, they can arbitrarily choose to stop rerolling after one or two rounds if they are happy with the combination they have already attained. Also, to make this more fun, you don't have to reroll all three dice, but rather you can reroll any number of the dice by your choice. However, only the final combination would be considered. This means when a player had (1,1,4) during the second round and was not satisfied with the "4", if the player choose to only reroll the singled out die and attained (1,1,2), this would be considered the player's final result, regardless of the fact that this player once had a better combination.

Also, players are not supposed to see other's dice during the three rounds of rolling process, so one cannot decide whether to reroll their dice by checking whether he or she is already ahead of their opponents. This is a game in which players themselves are completely responsible of whether they should gamble and risk the existing score they already had, in return of the possibility of a better combination.

A regular game would start by each player placing chips or tokens as their bet in the middle. Whoever wins the game at showdown would take the whole pot. However, just like Blackjack, a tie is possible, and when this happens, each player takes back their own bet, or as what is referred to as a chop-pot.

There are a few strategies that one can think of when playing this game. Following are 3 strategies that we choose to start our simulation with.

Strategy 1: Rerolling all three dice each round, unless hitting three-of-a-kind or four-

five-six.

Strategy 2: Stops rerolling if hitting three-of-a-kind or four-five-six. This strategy aims to hit four-five-six. It would reroll everything unless there are at least two dice from the combination (4,5,6), then it would only reroll the third die. For example, even if we hit (4,4,6), we would still reroll one of the 4s, with the aim to make it a 5.

Strategy 3: Stops rerolling if hitting three-of-a-kind or four-five-six. If already rolled a pair, we call the singled-out die' s face value n. If $n < 3$, we only reroll the singled-out die. If $n \geq 3$, we are happy with this combination and accept it as our final score, no matter if we still have rerolling chances left. This is rather a risk-averse strategy.

Instinctively, one would think that strategy 3 is much better than strategy 1, as in strategy 3 we treat special cases separately, instead of just rerolling everything hoping to hit some big combinations. Let player 1 plays the game with strategy 1, and player 2 with strategy 3. We will use this as an example to illustrate the details of the simulation. The following code is used to generate the simulations:

```
package simulation;
import java.util.*;
public class Simulation {
    public static void main(String[] args) {
        double[] winRate = new double[100];
        double[] tieRate = new double[100];
        for (int x = 0; x < 100; x++) {
            // double[] converge = new double[100];
            //strategy 1 with n >= 3
            int[] result1 = new int[10000];
            for (int i = 0; i < 10000; i++) {
                Random rand = new Random();
                //rolling three dices
                int roll1 = rand.nextInt(6) + 1;
                int roll2 = rand.nextInt(6) + 1;
                int roll3 = rand.nextInt(6) + 1;
                int[] rolls = new int[]{roll1, roll2, roll3};
                Arrays.sort(rolls);
                //initialize the score
                int score = 0;
                int count = 1;
                while (count <= 3) {
                    if (three_k(rolls)) {
                        score = 10 * rolls[1];
                        break;
                    } else if (two_k(rolls)) {
                        if (rolls[2] >= 3) {
                            score = rolls[2];
```

```
                    break;
                } else {
                    score = rolls[2];
                    rolls[2] = rand.nextInt(6) + 1;
                    count += 1;
                }
            } else if (ffs(rolls)) {
                score = 100;
                break;
            } else if (ott(rolls)) {
                score = -1;
                break;
            } else {
                rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                Arrays.sort(rolls);
                count += 1;
            }
        }
        result1[i] = score;
}
//System.out.println(Arrays.toString(result1));

//strategy 2 with n >= 5
int[] result2 = new int[10000];
for (int i = 0; i < 10000; i++) {
        Random rand = new Random();
        //rolling three dices
        int roll1 = rand.nextInt(6) + 1;
        int roll2 = rand.nextInt(6) + 1;
        int roll3 = rand.nextInt(6) + 1;
        int[] rolls = new int[]{roll1, roll2, roll3};
        Arrays.sort(rolls);
        //initialize the score
        int score = 0;
        int count = 1;
        while (count <= 3) {
            if (three_k(rolls)) {
                score = 10 * rolls[1];
                break;
            } else if (two_k(rolls)) {
                if (rolls[2] >= 5) {
                    score = rolls[2];
                    break;
                } else {
```

```
                    score = rolls[2];
                    rolls[2] = rand.nextInt(6) + 1;
                    count += 1;
                }
            } else if (ffs(rolls)) {
                score = 100;
                break;
            } else if (ott(rolls)) {
                score = -1;
                break;
            } else {
                rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                Arrays.sort(rolls);
                count += 1;
            }
        }
        result2[i] = score;
}
//System.out.println(Arrays.toString(result2));

//strategy 3 with the aim of three of a kind
int[] result3 = new int[10000];
for (int i = 0; i < 10000; i++) {
        Random rand = new Random();
        //rolling three dices
        int roll1 = rand.nextInt(6) + 1;
        int roll2 = rand.nextInt(6) + 1;
        int roll3 = rand.nextInt(6) + 1;
        int[] rolls = new int[]{roll1, roll2, roll3};
        Arrays.sort(rolls);
        //initialize the score
        int score = 0;
        int count = 1;
        while (count <= 3) {
            if (three_k(rolls)) {
                score = 10 * rolls[1];
                break;
            } else if (ffs(rolls)) {
                score = 100;
                break;
            } else if (ott(rolls)) {
                score = -1;
                break;
            } else if (two_k(rolls)) {
```

```
                score = rolls[2];
                rolls[2] = rand.nextInt(6) + 1;
                count += 1;
            } else {
                rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                Arrays.sort(rolls);
                count += 1;
            }
        }
        result3[i] = score;
}
//System.out.println(Arrays.toString(result3));

//strategy 4 with the aim of four-five-six
int[] result4 = new int[10000];
for (int i = 0; i < 10000; i++) {
        Random rand = new Random();
        //rolling three dices
        int roll1 = rand.nextInt(6) + 1;
        int roll2 = rand.nextInt(6) + 1;
        int roll3 = rand.nextInt(6) + 1;
        int[] rolls = new int[]{roll1, roll2, roll3};
        Arrays.sort(rolls);
        //initialize the score
        int score = 0;
        int count = 1;
        while (count <= 3) {
            if (three_k(rolls)) {
                score = 10 * rolls[1];
                break;
            } else if (ffs(rolls)) {
                score = 100;
                break;
            } else if (ott(rolls)) {
                score = -1;
                break;
            } else if (potential456(rolls)) {
                rolls[2] = rand.nextInt(6) + 1;
                Arrays.sort(rolls);
                count += 1;
            } else {
                rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                Arrays.sort(rolls);
                count += 1;
```

```
            }
        }
        result4[i] = score;
}
//System.out.println(Arrays.toString(result4));

//strategy 5: rerolling everything each round, the original cee-lo
int[] result5 = new int[10000];
for (int i = 0; i < 10000; i++) {
        Random rand = new Random();
        //rolling three dices
        int roll1 = rand.nextInt(6) + 1;
        int roll2 = rand.nextInt(6) + 1;
        int roll3 = rand.nextInt(6) + 1;
        int[] rolls = new int[]{roll1, roll2, roll3};
        Arrays.sort(rolls);
        //initialize the score
        int score = 0;
        int count = 1;
        while (count <= 3) {
            if (three_k(rolls)) {
                score = 10 * rolls[1];
                break;
            } else if (ffs(rolls)) {
                score = 100;
                break;
            } else if (ott(rolls)) {
                score = -1;
                break;
            } else if (two_k(rolls)) {
                if (rolls[2] >= 3) {
                    score = rolls[2];
                    break;
                } else {
                    rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                    count += 1;
                }
            } else {
                rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                Arrays.sort(rolls);
                count += 1;
            }
        }
        result5[i] = score;
```

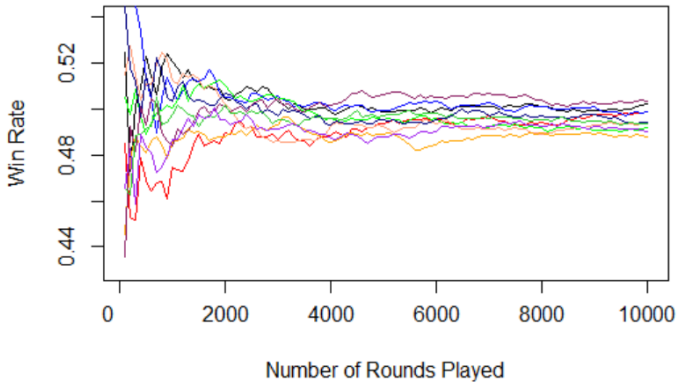
```
}
//System.out.println(Arrays.toString(result5));

//strategy 6: combining previous strategies
int[] result6 = new int[10000];
for (int i = 0; i < 10000; i++) {
        Random rand = new Random();
        //rolling three dices
        int roll1 = rand.nextInt(6) + 1;
        int roll2 = rand.nextInt(6) + 1;
        int roll3 = rand.nextInt(6) + 1;
        int[] rolls = new int[]{roll1, roll2, roll3};
        Arrays.sort(rolls);
        //initialize the score
        int score = 0;
        int count = 1;
        while (count <= 3) {
                if (three_k(rolls)) {
                        score = 10 * rolls[1];
                        break;
                } else if (ffs(rolls)) {
                        score = 100;
                        break;
                } else if (ott(rolls)) {
                        score = -1;
                        break;
                } else if (two_k(rolls)) {
                        if (rolls[2] >= 4) {
                                score = rolls[2];
                                break;
                        } else {
                                score = rolls[2];
                                rolls[2] = rand.nextInt(6) + 1;
                                count += 1;
                        }
                } else if (potential456(rolls)) {
                        rolls[2] = rand.nextInt(6) + 1;
                        Arrays.sort(rolls);
                        count += 1;
                } else {
                        rolls = new int[]{rand.nextInt(6) + 1, rand.nextInt(6) + 1, rand.nextInt(6) + 1};
                        Arrays.sort(rolls);
                        count += 1;
                }
```

```
            }
            result6[i] = score;
        }
        //System.out.println(Arrays.toString(result6));

        /*
        for(int k = 100; k <= 10000; k += 100){
            int winA = 0;
            int ties = 0;
            for (int i = 0; i < k; i++) {
                if (result1[i] > result5[i]) {
                    winA++;
                } else if (result1[i] == result5[i]) {
                    ties++;
                }
            }
            converge[k/100-1] = (double) winA / (k+1);
        }

        System.out.println(Arrays.toString(converge));
         */
        int winA = 0;
        int ties = 0;
        for (int i = 0; i < 10000; i++) {
            if (result1[i] > result5[i]) {
                winA++;
            } else if (result1[i] == result5[i]) {
                ties++;
            }
        }

        //System.out.println((double) winA/10000);
        //System.out.println((double) ties/10000);
        winRate[x] = (double) winA / 10000;
        tieRate[x] = (double) ties / 10000;

    }

    double maxWin = winRate[0];
    double minWin = winRate[0];
    double maxTie = tieRate[0];
    double minTie = tieRate[0];

    for (int i = 1; i < 100; i++) {
```

```
            if (winRate[i] > maxWin) {
                maxWin = winRate[i];
            }
            if (winRate[i] < minWin) {
                minWin = winRate[i];
            }
            if (tieRate[i] > maxTie) {
                maxTie = tieRate[i];
            }
            if (tieRate[i] < minTie) {
                minTie = tieRate[i];
            }
        }
        //System.out.println(Arrays.toString(winRate));
        //System.out.println(Arrays.toString(tieRate));
        System.out.println(minWin + " | " + maxWin);
        System.out.println(minTie + " | " + maxTie);
    }

    // check if the rolls are three of a kind
    public static boolean three_k(int[] rolls) {
        if (rolls.length == 3 && rolls[0] == rolls[1] && rolls[1] == rolls[2]) {
            return true;
        }
        return false;
    }

    //check if the rolls are two of a kind, and sort them
    public static boolean two_k(int[] rolls) {
        if (rolls.length == 3 && (rolls[0] == rolls[1] || rolls[1] == rolls[2])) {
            if (rolls[1] == rolls[2]) {
                int temp = rolls[0];
                rolls[0] = rolls[1];
                rolls[1] = rolls[2];
                rolls[2] = temp;
            }
            return true;
        }
        return false;
    }

    //check 456
    public static boolean ffs(int[] rolls) {
        if (rolls[0] == 4 && rolls[1] == 5 && rolls[2] == 6) {
```

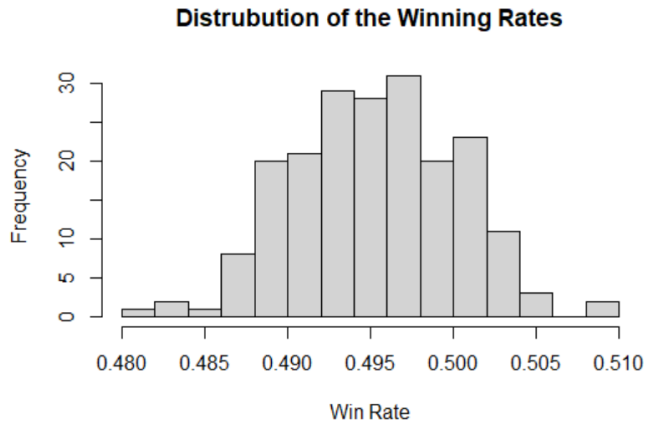
```
            return true;
        }
        return false;
    }

    //check 123
    public static boolean ott(int[] rolls) {
        if (rolls[0] == 1 && rolls[1] == 2 && rolls[2] == 3) {
            return true;
        }
        return false;
    }

    public static boolean potential456(int[] rolls) {
        if (!ffs(rolls)) {
            List temp = Arrays.asList(rolls);
            if (temp.contains(4) && temp.contains(5)) {
                rolls[0] = 4;
                rolls[1] = 5;
                return true;
            } else if (temp.contains(4) && temp.contains(6)) {
                rolls[0] = 4;
                rolls[1] = 6;
                return true;
            } else if (temp.contains(5) && temp.contains(6)) {
                rolls[0] = 5;
                rolls[1] = 6;
                return true;
            }
        }
        return false;
    }

}
```

We ran the simulation first from 100 times, all the way to 10,000 times, each time with an additional 100 increment. And we plot the winning rate of strategy 3 at each increment against the total number of times we ran the simulation. Suppose we have 10 of such lines, meaning we run the whole thing 10 times, each time with 10,000 simulation trials, and each time calculating strategy 3' s winning rate when we are at 100, 200, ⋯, 9,900, and 10,000 trials. We graph these 10 lines in R studio,



We can see from the graph very straight-forwardly that the more times we simulate the game, the more accurate our estimation of strategy 2' s winning rate is. As the 10 lines tend to gather around a narrow interval in the end, we have some confidence that our simulation is functional.

Furthermore, by the Central Limit Theorem, the randomly generated win rates should be approximately normally distributed. To have a more visual conclusion, following is the histogram of 200 winning rates of strategy 3, each calculated from 10,000 trials,



Indeed, the distribution of these win rates is very nearly normal. Therefore, if we take a closer look at the first 10 values and find that the minimum is 0.4893, while the maximum being 0.5004. Then, as the central limit theorem says they are approximately normally distributed, they should be symmetric about the mean, hence we are 99.8% confidence that the true win rate lies between the interval [0.4893, 0.5004].

Using the same method, we are 99.8% confidence that the true tie rate lies between the interval [0.1189, 0.135]

Hence, we can conclude that strategy 3 would win the game around half of the time, tying with player who uses strategy 1 roughly one-tenths of the time. In other words, on average strategy 3 wins the game roughly 10% percent more frequently than strategy 1. So,

our instinct is correct that rerolling everything each time is not a wise move, and that employing strategy 3 is indeed a better approach to win the game.

The same logic applies when we use strategy 3 to play against strategy 2. We find that we are 99.8% confidence that strategy 3' s true win rate is between [0.7125, 0.7177] and the true tie rate is between [0.1024, 0.1054]. This is to say that strategy 3 is by far the best strategy to win cee-lo in those three strategies.

Now, based on strategy 3, we try to make a few modifications and see if they would further increase our win rate.

First, what would happen if we are willing to take more risks after attaining a pair of dice. Maybe we can keep rerolling the third dice with the aim to reach three of a kind. We call this strategy 4.

Strategy 4: Stops rerolling if hitting three-of-a-kind or four-five-six. If already rolled a pair, we continue to reroll the singled-out die, regardless of the value of n, with the aim to hit three of a kind. This is rather a risk-taking strategy.

Furthermore, strategy has a n value of 3. To reiterate, the value of n stands for the singled-out die' s face value after rolling a pair. Now, with all other details being the same as strategy 3, we try to increase the value of n, say all the way to n=6. We have our strategy 5 to 7.

Strategy 5: Stops rerolling if hitting three-of-a-kind or four-five-six. This time we are only happy if after rolling a pair, $n \geq 4$. We reroll the singled-out die if otherwise.

Strategy 6: With everything else being exactly the same as strategy 5, but let $n \geq 5$.

Strategy 7: With everything else being exactly the same as strategy 5, but let $n \geq 6$.

We let all these 4 strategies play against each other, detailed results would be presented later in charts. Here, we find that strategy 6 seems to prevail when played against any other new strategies and its predecessor strategy 3. All with 99.8% confidence, when strategy 6 is used to play with strategy 3, its win rate is between [48.97, 49.62] while the tie rate is between [10.4, 11.08], meaning it wins roughly 10% more times than the original strategy 3.

With the same logic, when up against strategy 4, Strategy 6' s win rate is between [47.89, 49.53] while tie rate is between [8.06, 8.83], meaning it wins roughly 5% more times than its opponent. When up against strategy 5, strategy 6' s win rate is between [45.73, 46.93] and tie rate is between [10.79, 11.47], meaning it wins roughly about 4% more. Lastly, if we have strategy 5 play against strategy 7, the win rate is between [44.01, 46.06] and tie rate is between [9.79, 10.39], this time only winning about 1% more times.

Hence, we can conclude that setting $n \geq 3$ is too risk-averse while to have $n \geq 6$ is too risk-tolerance. The optimum n value would be 5.

Now that we know strategies work best when we isolate the pair when we have it and reroll the singled-out die if its face value is less than 5, we try to further modify this strategy by combining it with strategy 2.

Strategy 8: A combined strategy. Stops rerolling if hitting three-of-a-kind or four-five-six. If we have a pair, we would act like we did in the best strategy so far, strategy 6, with setting $n \geq 5$. Meanwhile, if we only have two of (4,5,6), we reroll the third die to aim for four-five-six. However, the pair strategy would have a higher priority here. This is to say if we have (5,5,6), instead of rerolling one of the 5s, we would stop and use this current combination

as our final result.

Have this new strategy play against our strategy 6, we find that we are 99.8% confidence that strategy 8' s win rate is between [43.44, 44.55], loss rate between [43.61, 45.07], and tie rate between [11.02, 12.1]. The difference here is too insignificant for us to draw and meaningful conclusion. So, we think this last modification does not improve our win rate.

The following chart summarizes all the game results between different strategies. In each cell, the first value is the win rate and the second one is the tie rate. Each of these results was attained by running the simulation 10,000 times.

	1	2	3	4	5	6	7	8
1	[41.28, 42.27] [41.05, 42.71] [15.96, 16.71]	[57.96, 60.2] [17.6, 18.97] [22.2, 23.57]	[36.9, 38.5] [48.56, 50.45] [12.14, 13.27]	[37.54, 39.52] [50.73, 52.42] [9.04, 10.04]	[34.46, 36.12] [51.83, 53.81] [11.47, 12.3]	[33.6, 35.6] [53.34, 55.16] [11.06, 11.72]	[34.83, 36.66] [52.96, 54.83] [10.06, 10.85]	[33.91, 35.36] [53.3, 54.91] [10.66, 11.89]
2	[17.6, 18.97] [57.96, 60.2] [22.2, 23.57]	[18.93,20.19] [18.68, 20.2] [59.8, 62.29]	[17.51, 18.58] [70.79, 72.52] [9.97, 11.16]	[16.97, 17.7] [71.7, 72.63] [10.12, 10.97]	[17.31, 18.37] [71.17, 72.14] [10.33, 11.08]	[16.75, 18.24] [71.26, 72.34] [10.19, 10.91]	[16.83, 18.68] [70.75, 72.61] [10.3, 11.22]	[17.01, 18.1] [71.35, 72.79] [9.89, 10.76]
3	[48.56, 50.45] [36.9, 38.5] [12.14, 13.27]	[70.79, 72.52] [17.51, 18.58] [9.97, 11.16]	[43.03, 44.52] [43.55, 45.06] [11.66, 12.5]	[44.69, 46.48] [44.97, 46.32] [8.48, 9.03]	[40.4, 42.14] [46.07, 48.34] [11.14, 12.02]	[39.77, 40.3] [48.97, 49.62] [10.4, 11.08]	[40.75, 42.43] [48.05, 49.51] [9.27, 10.16]	[39.01, 40.89] [48.51, 50.31] [9.94, 10.93]
4	[50.73, 52.42] [37.54, 39.52] [9.04, 10.04]	[71.7, 72.63] [16.97, 17.7] [10.12, 10.97]	[44.97, 46.32] [44.69, 46.48] [8.48, 9.03]	[45.36, 46.15] [45.65, 46.65] [7.91, 8.35]	[42.94, 44.2] [46.73, 47.98] [8.13, 9.16]	[42.31, 43.38] [47.89, 49.53] [8.06, 8.83]	[42.77, 44.18] [47.38, 48.72] [8.17, 8.76]	[42.45, 43.85] [47.33, 49.06] [8.12, 8.89]
5	[51.83, 53.81] [34.46, 36.12] [11.47, 12.3]	[71.17, 72.14] [17.31, 18.37] [10.33, 11.08]	[46.07, 48.34] [40.4, 42.14] [11.14, 12.02]	[46.73, 47.98] [42.94, 44.2] [8.13, 9.16]	[43.23, 44.73] [43.34, 44.92] [11.76, 12.35]	[41.69, 43.17] [45.73, 46.93] [10.79, 11.47]	[42.59, 43.84] [45.78, 47.63] [9.76, 10.38]	[41.31, 42.72] [45.8, 47.02] [10.7, 11.68]
6	[53.34, 55.16] [33.6, 35.6] [11.06, 11.72]	[71.26, 72.34] [16.75, 18.24] [10.19, 10.91]	[48.97, 49.62] [39.77, 40.3] [10.4, 11.08]	[47.89, 49.53] [42.31, 43.38] [8.06, 8.83]	[45.73, 46.93] [41.69, 43.17] [10.79, 11.47]	[43.69, 44.69] [43.68, 45.11] [11.16, 11.98]	[44.01, 46.06] [43.98, 45.6] [9.79, 10.39]	[43.61, 45.07] [43.44, 44.55] [11.02, 12.1]
7	[52.96, 54.83] [34.83, 36.66] [10.06, 10.85]	[70.75, 72.61] [16.83, 18.68] [10.3, 11.22]	[48.05, 49.51] [40.75, 42.43] [9.27, 10.16]	[47.38, 48.72] [42.77, 44.18] [8.17, 8.76]	[45.78, 47.63] [42.59, 43.84] [9.76, 10.38]	[43.98, 45.6] [44.01, 46.06] [9.79, 10.39]	[44.18, 45.69] [44.25, 45.58] [9.68, 10.83]	[44.16, 45.66] [44.18, 45.68] [9.6, 10.52]
8	[53.3, 54.91] [33.91, 35.36] [10.66, 11.89]	[71.35, 72.79] [17.01, 18.1] [9.89, 10.76]	[48.51, 50.31] [39.01, 40.89] [9.94, 10.93]	[47.33, 49.06] [42.45, 43.85] [8.12, 8.89]	[45.8, 47.02] [41.31, 42.72] [10.7, 11.68]	[43.44, 44.55] [43.61, 45.07] [11.02, 12.1]	[44.18, 45.68] [44.16, 45.66] [9.6, 10.52]	[43.7, 44.91] [43.56, 45.03] [11.05, 12.0]

Win Rate Interval (%)
Loss Rate Interval (%)
Tie Rate Interval (%)

Conclusions and findings:

There are a few things worth noticing here. First, if we look at the diagonal cells in the above chart, we can see that when two players are playing the game following the same strategy, such as roll 2 column 2 means both players are using strategy 2, their win rate is roughly the same every time. This should suit one' s instinct and prove the game' s fairness, as playing it for a large number of times, in our case is 10×10,000 times, two players with the same strategy should win the game an almost equal number of times.

Now, the worst two is strategy 1 and strategy 2. We have talked about how strategy 1 is not wise as it rerolls everything unless it hits a big combination. To dive into this even

further, one can see that with 3 six-sided dice, we have 216 possible combinations in total. Therefore, if we were to reroll everything every time, hitting a four-five-six has only a 6/218 possibility, or 2.7778%, same as hitting three of a kind and an instant loss of one-two-three. Another 41.67% percent of the time it would have a pair and a single point, while the rest of the 50% of times it would just be random and meaningless combinations.

However, strategy 2 is apparently even worse, as it has the lowest win rate, and an oddly high tie rate when played against itself. The low win rate is due to its risk tolerance. By aiming for a four-five-six, it forsakes the score it could have already achieved. For example, if a player with strategy 5 had (4,4,6) after the second round, the player would reroll one of the 4s. In this scenario, only one sixths of the time would the player reach a four-five-six, and another one-third of the time it would have (4,4,5) or (6,6,4), at least attaining some combination. However, this means for half of the time the player would end up with meaningless combination such as (1,4,6), and thus forsaking the score he or she could have had and displaying a score of 0 during final showdown. This also explains the extraordinarily high tie rates when played against itself, for there are too many cases in which two players both with strategy 2 would end up with a score of 0 at the same time.

We though that strategy 8, being the most complex and thorough, would have the highest win rate. However, our strategy 6, 7, and 8' s data came back too similar. So, we increase the number trials in each simulation to 100,000, and with a total number 100 simulations. The win, loss, tie rate' s confidence interval of strategy 6 vs. 7 came back as [44.33, 45.097], [44.775, 45.521], and [9.819, 10.353]. The win, loss, tie rate' s confidence interval of strategy 6 vs. 8 came back as [43.771, 44.612], [43.789, 44.66], and [11.357, 11.812]. The win, loss, tie rate' s confidence interval of strategy 7 vs. 8[44.575, 45.505], [44.322, 45.224], and [9.929, 10.475]. Since their difference is minute and all overlaps with each other, we wouldn' t say they have too much statistical significance. best.

If we have to make a decision, our strategy 6 came out relatively better than others, as it has the relatively highest win rate against every other strategy. Hence, leaving the rolled pairs untouched, and only rerolling the singled-out die when its face value is less than 5 appears to be the best strategy to win a cee-lo heads up.