

Solitario di Prina

Gallina Roberto

13/11/2022

Contents

1	Premesse	2
2	Funzionamento del gioco	3
2.1	Sequenza corretta	4
2.2	Sequenza perfetta	4
2.3	Sequenza n-perfect	5
3	Introduzione	6
4	Analisi statistica	6
5	Implementazione	7
6	Conclusioni	10
7	Dati statistici	11

1 Premesse

In tutto il documento ci si riferirà a il *mazzo* (*Deck*), esso è da intendere come un mazzo di 40 carte, divise in 4 semi. Per convenzione durante il documento si userà il mazzo francese le cui carte sono:

- l'asso
- il due
- il tre
- il quattro
- il cinque
- il sei
- il sette
- il fante (J)
- la regina (Q)
- il re (K)

Mentre i semi sono:

- i cuori
- i quadri
- i fiori
- i picche

Il mazzo è quindi composta dalle carte sottostanti:









































									
									
									
									

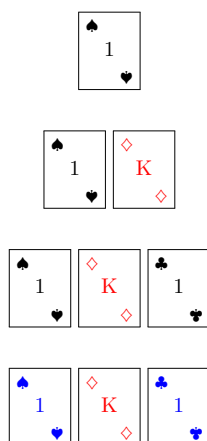
Figure 1: Mazzo

In tutto il documento ci si riferirà a la *sequenza*, essa rappresenta l'ordine delle carte nel mazzo mescolato.

2 Funzionamento del gioco

Il gioco è molto semplice, dato un mazzo da gioco mischiato, si tengono tutte le carte coperte in pila; si scopre le prime tre carte. Nel caso la prima e la terza carta hanno stesso valore o stesso seme, allora la seconda carta viene spostata sopra la prima (avvicinando la terza). Successivamente si scopre un'altra carta di ricomincia a controllare dall'inizio.

Ecco un esempio del funzionamento



Avendo lo stesso valore (1), la carte centrale sale sopra la prima

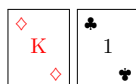


Figure 2: Esempio di funzionamento

2.1 Sequenza corretta

Viene definita *sequenza corretta*, una sequenza, in cui terminate le carte si hanno due esattamente due pile di carte.

Eccone un esempio

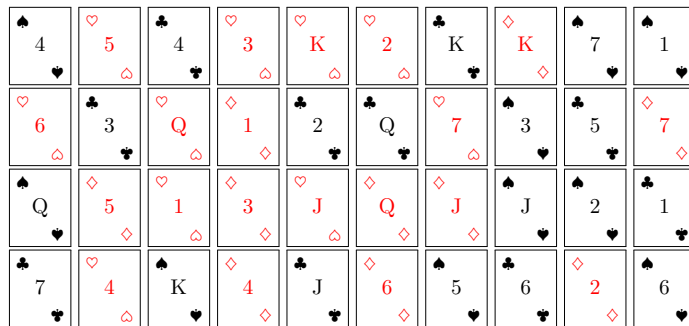


Figure 3: Sequenza corretta

2.2 Sequenza perfetta

Viene definita *sequenza perfetta*, una sequenza corretta, in cui è esattamente l'ultima carta a formare la seconda pila; per cui terminate le carte di hanno esattamente due pile di carte in cui la seconda pila ha esattamente una carta

Eccone un esempio

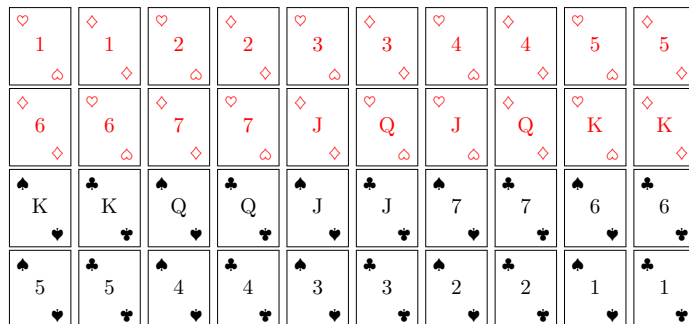


Figure 4: Sequenza perfetta

2.3 Sequenza n-perfect

Viene definita *sequenza n-perfect*, una sequenza, in cui terminate le carte non ci sono pile con più di un carta, ossia il numero delle pile è uguale al numero delle carte

Eccone un esempio

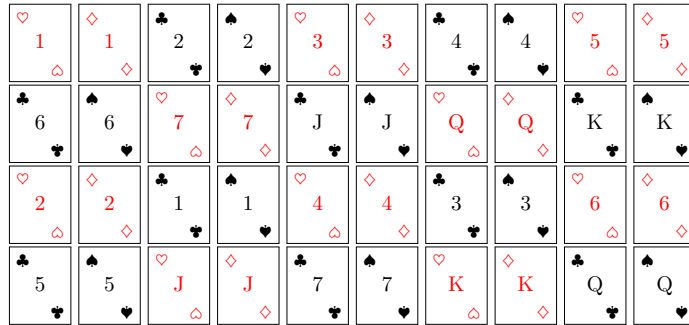


Figure 5: Esempio di sequenza n-perfect

3 Introduzione

Durante le innumerevoli partite giocate da me giocate, non è capitato che la sequenza fosse corretta o ben che meno perfetta, e quindi nata in me l'idea di sapere quanto siano rare tali sequenze, ho quindi iniziato ad analizzare il gioco dal punto di vista statistico.

Poichè il mazzo è composto da 40 carte, possiamo calcolare il numero totale di sequenze, esso sarà pari a $40! \simeq 8.1591528 \cdot 10^{47}$; questo numero computazionalmente enorme, non è quindi possibile su un computer verificare tutte le sequenze possibili per avere dati esatti. Allo stesso modo anche calcolare quante siano le sequenze corrette, perfette o n-perfect non è semplice, avendo dati esempi di tutte e tre possiamo sicuramente affermare che ne esistano.

4 Analisi statistica

Definiamo P_C la probabilità che una sequenza sia corretta, P_P la probabilità che una sequenza sia perfetta e P_{N-P} la probabilità che una sequenza sia n-perfect; queste probabilità sono a noi sconosciute, ma sicuramente $P_C > 0$, $P_P > 0$ e $P_{N-P} > 0$ avendo dato un esempio per ognuna.

Ci aspettiamo che su 10 sequenze, le sequenze corrette S_C^{10} siano:

$$S_C^{10} \simeq 10 \cdot P_C$$

Possiamo quindi isolare P_C

$$P_C \simeq \frac{S_C^{10}}{10}$$

Ovviamente questa è un'approssimazione, ma aumentando il numero di sequenze otterremo un risultato sempre più vicino a P_C , possiamo quindi definire:

$$P_C = \lim_{n \rightarrow +\infty} \frac{S_C^n}{n}$$

Allo stesso modo definiamo:

$$P_P = \lim_{n \rightarrow +\infty} \frac{S_P^n}{n} \quad P_{N-P} = \lim_{n \rightarrow +\infty} \frac{S_{N-P}^n}{n}$$

Possiamo quindi scrivere un programma per generare quante più sequenze possibili in modo approssimare queste probabilità.

5 Implementazione

Iniziamo col definire gli enumeratori per i semi e per i valori, in modo da poterli usare con semplicità dopo.

```
1 class Seme {
2     public static HEART = "1";
3     public static DIAMOND = "2";
4     public static FLOWER = "3";
5     public static CLUB = "4";
6 }
```

```
1 class Value {
2     public static ONE = "1";
3     public static TWO = "2";
4     public static THREE = "3";
5     public static FOUR = "4";
6     public static FIVE = "5";
7     public static SIX = "6";
8     public static SEVEN = "7";
9     public static JACK = "J";
10    public static QUEEN = "Q";
11    public static KING = "K";
12 }
```

Definiamo ora una carta come una classe che memorizza un seme e un valore.

```
1 class Card{
2     private Seme seme;
3     private Value value;
4
5     constructor(seme, value) {
6         this.seme = seme;
7         this.value = value;
8     }
9
10    public Seme getSeme() {
11        return this.seme
12    }
13
14    public Value getValue() {
15        return this.value
16    }
17 }
```

Definiamo poi il mazzo come un contenitore di 40 carte; definiamogli poi:

- un metodo *shuffle* che randomicamente mischia le carte nel mazzo.
- un metodo *sameValueOrSeme* che confronta due carte, ritornando *true* se hanno lo stesso valore o lo stesso seme
- un metodo *check* che data una sequenza di carte le confronta facendo salire quelle con lo stesso valore o seme, ritornando la sequenza ridotta
- un metodo *isCorrect* che analizza la sequenza e dice se la sequenza ridotta è di 2 carte

- un metodo *isPerfect* che analizza la sequenza e dice se la sequenza ridotta è di 2 carte (in cui l'ultima è esattamente l'ultima scesa)
- un metodo *isNPerfect* che analizza la sequenza e dice se la sequenza ridotta è di 40 carte

La classe risultante sarà simile a questa

```
1  class Deck{
2      private Card cards[40];
3
4      constructor() {
5          this.cards.add(new Card(Seme.HEART, Value.ONE));
6          this.cards.add(new Card(Seme.DIAMOND, Value.ONE));
7          .
8          .
9      }
10
11     public void shaffle() {
12         cards.randomSort();
13     }
14
15     public bool sameValueOrSeme(a, b) {
16         return a.value == b.value || a.seme == b.seme;
17     }
18
19     public Card[] check(cards) {
20         i = 0;
21         j = 2;
22
23         while (j < cards.length) {
24             if (sameValueOrSeme(cards[i], cards[j])) {
25                 cards.removeByIndex(i);
26                 i = 0;
27                 j = 2;
28             } else {
29                 i++;
30                 j++;
31             }
32         }
33         return cards;
34     }
35
36     public bool isCorrect() {
37         return check(cards).length == 2;
38     }
39
40     public bool isPerfect() {
41         last = cards.pop();
42         cards = check(cards);
43
44         if (cards.length == 2) {
45             return sameValueOrSeme(cards[0], last);
46         }
47         return false;
48     }
49
50     public bool isNPerfect() {
51         return check(cards).length == 40;
52     }
53 }
```

Possiamo quindi scrivere un programma per calcolare sequenze corrette, perfette o n-perfect.

```

1  do {
2      d = new Deck();
3      d.shuffle();
4  } while (d.isCorrect());
5  // or d.isPerfect() or d.isNPerfect()
6
7  d.print(); // is a method for print the sequence

```

Oppure, ben più interessante, possiamo generare diverse sequenze contare quante di esse siano corrette, perfette o n-perfect

```

1  const MAX_TENT = 1 * 1000 * 1000;
2  correct = 0;
3  perfect = 0;
4  n_perfect = 0;
5
6  for (i = 0; i < MAX_TENT; i++) {
7      d = new Deck();
8      d.shuffle();
9
10     if (d.isCorrect()) {
11         correct++;
12         if (d.isPerfect())
13             perfect++;
14     }
15     if (d.isNPerfect())
16         n_perfect++;
17 }

```

Possiamo ora lanciare il programma generando 10, 100, 1 000, 10 000, 1 000 000, 10 000 000 e 100 000 000 sequenze; vediamo il risultato

Sequenze generate	Sequenze corrette	P_C	Sequenze perfette	P_P	Sequenze n-perfect	P_{NP}
10	1	0.1	0	0	0	0
10^2	2	2	0	0	0	0
10^3	5	0.5	1	0.1	0	0
10^4	42	0.42	6	0.6	0	0
10^5	456	0.4559	141	0.141	0	0
10^6	4607	0.4607	1431	0.1431	1	0.0001
10^7	46245	0.46245	13995	0.13995	9	0.00009
10^8	462679	0.462679	140521	0.140521	86	0.000086
10^9	4	0.462679	1	0.140521	1	0.000086

Table 1: Risultati delle approssimazioni

6 Conclusioni

Come era prevedibile la generazione di più sequenze ha portato a un'approssimazione sempre migliore, possiamo quindi presumere che:

$$P_C \simeq 0.0 \qquad P_P \simeq 0.0 \qquad P_{N-P} \simeq 0.0$$

Inoltre possiamo anche stimare il numero totale di sequenze corrette S_C

$$S_C = 40! \cdot P_C \simeq 0.0$$

il numero totale di sequenze perfette S_P

$$S_P = 40! \cdot P_P \simeq 0.0$$

e il numero totale di sequenze n-perfect S_{N-P}

$$S_{N-P} = 40! \cdot P_{N-P} \simeq 0.0$$

Inoltre come da definizione, ogni sequenza perfetta è una sequenza corretta, ma non vale il contrario, esistono quindi delle sequenze corrette che non sono perfette, definiamo quindi

7 Dati statistici

Possiamo poi modificare leggermente lo script, infatti usando il metodo check, possiamo valutare la lunghezza della sequenza ridotta e ottenere più dati riguardo la distribuzione delle sequenze.

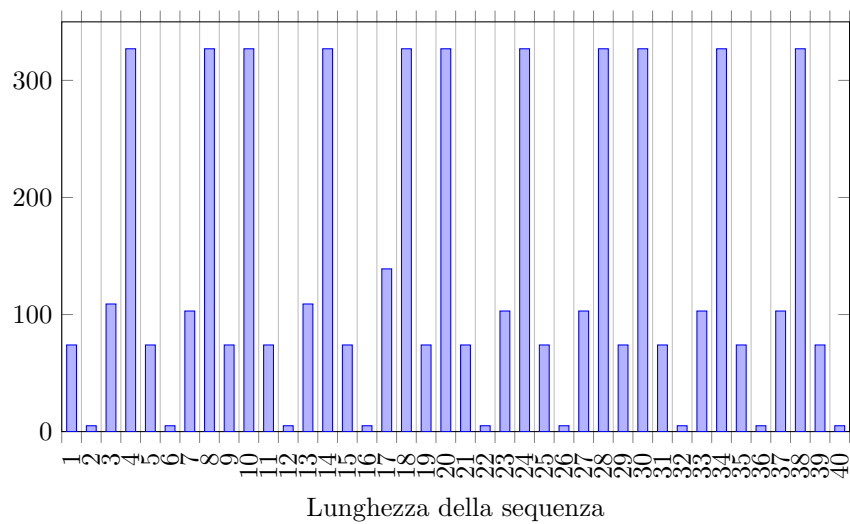


Figure 6: Distribuzione statistica delle sequenze

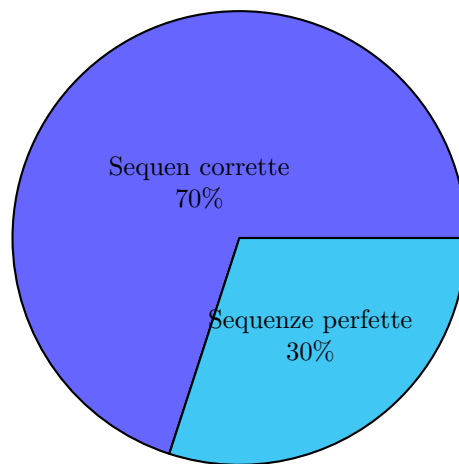


Figure 7: Sequenze perfette rispetto alle sequenze corrette