Paper 88-2010

Simulating a Simple Solitaire Card Game Using SAS®

Robert Moore, Ameriprise Financial, Minneapolis, MN

ABSTRACT

This paper illustrates using SAS arrays and nested DO loops in a DATA step to simulate a card game. A SAS program is presented that simulates a simple version of solitaire played with a standard deck of cards. Arrays are used to represent the standard deck of playing cards, and a DATA step with nested DO loops is used to repeatedly play the game and capture the outcomes of each game. PROC FREQ and PROC GCHART are used to examine the distribution of possible outcomes of the game generated by simulating a large number of trials.

INTRODUCTION

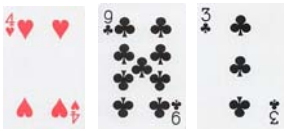
Solitaire refers to card games played with a standard deck of 52 cards that involve only a single player. This paper presents a SAS program to play one simple version of the game sometimes called “Simple Solitaire” or “Idiot’s Delight”. We assume a standard deck of 52 playing cards with the 4 suits of hearts, clubs, diamonds, and spades, and the 13 ranks of 2 through 10, jack, queen, king, and ace.

The game is played by dealing cards face up from a shuffled deck into the single players hand (which is most easily visualized as a line of cards from left to right). When the suit of the rightmost card in the hand matches the suit of the card located 3 cards to the left, then the intervening two cards are discarded and removed from play. When the rank of the rightmost card in the hand matches the rank of the card located 3 cards to the left, then all four cards are discarded and removed from play. The object of the game is to be left with as few cards in the hand as possible.

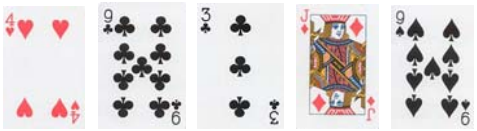
As an illustration, say the first four cards dealt are the 4 of hearts, the 9 of clubs, the queen of diamonds, and the 6 of spades, denoted as follows: 4H, 9C, QD, 6S.

Since the rightmost card (the 6 of spades) does not match either the suit or the rank of the card located 3 cards to the left (the 4 of hearts), another card is dealt. Say the next card is the 3 of clubs, so the hand looks as follows: 4H, 9C, QD, 6S, 3C.

Since the suit of the rightmost card (the 3 of clubs) matches the suit of the card located 3 cards to the left (the 9 of clubs), the intervening two cards are discarded and the hand now looks as follows: 4H, 9C, 3C.



Say the next 2 cards dealt are the jack of diamonds and the 9 of spades, so the hand looks as follows: 4H, 9C, 3C, JD, 9S.



Since the rank of the rightmost card (the 9 of spades) matches the rank of the card located 3 cards to the left (the 9 of clubs), all 4 cards are discarded, and the hand contains just 1 card (the 4 of hearts). Additional cards are dealt until there is either a rank or suit match with the card located 3 cards to the left of the rightmost card. The game continues until there are no more cards to be dealt, and the number of cards that remain in the hand is the score for that game.

It is of interest to determine the distribution of the number of cards left in the hand at the end of each game played. This can be explored by simulating a large number of games and recording the score (the number of cards left in the hand) after each game played. Note that there can be at most 52 cards left in the hand and in the best case no cards left in the hand. Also, since there are always either 2 or 4 cards discarded from the hand, there can only be an even number of cards left in the hand. So the distribution can only take the even values from 0 through 52.

PROGRAM TO SIMULATING THE CARD GAME

The program uses an array of 52 elements to represent the deck of cards, with each array element being a string of 2 characters. The first character is the rank of the card, and the second character is the suit of the card. So the first character can take the 13 possible rank values 2, 3, 4, 5, 6, 7, 8, 9, T, J, Q, K, and A, where the letters T, J, Q, K, and A are used to denote the 10, jack, queen, king, and ace respectively. The second character can take the 4 possible suit values H, C, D, and S to denote the hearts, clubs, diamonds, and spades.

The complete program is shown in figure 1 with line numbers to facilitate the discussion of the code. Line 1 defines the macro variable that is the number of time the game will be played. Lines 4 through 15 create a data set named 'gamedata' which will contain one observation which is the deck of 52 cards. Lines 5 and 6 define an array of 52 elements with each element being a 2 character string. Lines 8 through 14 initialize the deck of cards using a pair of nested DO loops. The outer DO loop cycles through the 4 possible suits and the inner DO loop cycles through the 13 possible ranks. Line 11 concatenates the 1 character rank and the 1 character suit and assigns the resulting 2 character string to the i^{th} element of the array which represents the i^{th} card in the deck. At this point the data set named 'gamedata' contains one observation which is the 52 element array or equivalently, the 52 different variables card1, card2, … card52 which have the values 2H, 3H, 4H, … KS, AS, respectively.

Now that the deck of cards has been initialized, and is contained in the data set named 'gamedata', the next DATA step on lines 17 through 56 plays the specified number of games and outputs the results of each game (the number of cards remaining in the hand) to the data set named 'games'.

Lines 19 and 20 define the array 'card' as the 52 variables card1 through card52 which was already initialized in lines 8 through 15. Lines 21 and 22 define the array 'hand' as the 52 variables hand1 through hand52 where each element of the array is a card (a 2 character string with the first character being the rank and the second character being the suit).

The DO loop that starts on line 24 and ends on line 53 plays the number of games specified in the macro variable 'num_games' assigned on line 1. Within this main DO loop, lines 26 through 31 contain a DO loop that shuffles the deck of cards. On line 27 a random integer between 1 and 52 is generated and assigned to the variable 'random_position'. The expression on the right side of the equation uses the RANUNI function to generate a random number from the uniform distribution on the interval 0 to 1. Providing the argument of 0 to the RANUNI function generates a different seed each time the function is executed. For testing and de-bugging the program it is convenient to use a positive number like 123 as the argument so that the same sequence of random numbers is generated each time the function is executed. Lines 28 through 30 just swap the i^{th} card in the deck with the card in the random position.

Lines 33 through 52 play the game by dealing cards from the deck into the hand and comparing the rightmost card in the hand with the card located 3 cards to the left of the rightmost card. The variable t contains the position of the rightmost card in the hand. It is initialized to 0 on line 33, and incremented or decremented as the game is played within the DO loop that starts on line 34 and ends on line 50. The DO-WHILE loop on lines 38 through 49 plays the game while the variable 'playing' (initialized to 1 or 'true' on line 37) remains 'true', and there are at least 4 cards in the hand (the variable t contains the number of cards in the hand).

The IF-THEN-ELSE clause on lines 39-43 checks if the rightmost card in the hand is the same suit as the card located 3 cards to the left of it, and if so, the two intervening cards are discarded. The position of the rightmost card in the hand is decremented by subtracting 2 from the variable t. The rest of the IF-THEN-ELSE clause on lines 45-48 checks if the rightmost card in the hand is the same rank as the card located 3 cards to the left of it, and if so, the four cards are discarded. The position of the rightmost card in the hand is decremented by subtracting 4 from the variable t. The SUBSTR function is used on lines 39 and 45 to compare the 2^{nd} character (the suit) or the 1^{st} character (the rank) of the cards.

If neither condition for discarding is met, then the variable ‘playing’ is set to 0 or ‘false’ on line 47, and the DO-WHILE loop ends and the program execution returns to the outer DO loop starting on line 34. Line 35 increments the position of the rightmost card in the hand, and line 36 deals one additional card from the deck to the hand. The variable ‘playing’ is reset to 1 or ‘true’ on line 37, and the DO-WHILE loop executes again until there is no condition for discarding.

When all the cards in the deck have been dealt to the hand, the DO loop on lines 34-50 ends, and on line 51 the variable cards_left is assigned the value of variable t which is the position of the rightmost card in the hand (the number of cards remaining). Line 52 outputs this value to the data set ‘games’, and execution returns to the main DO loop on lines 24 through 53, and another game is played. The keep statement on line 55 keeps only the two variables ‘game_number’ and ‘cards_left‘ in the data set ‘games’ when the simulation is complete.

TESTING

In developing the program, testing was performed by capturing the deck (the array of 2 character variables card1 through card52) in a data set and reviewing the output. For example, the card shuffle section of the code was tested by generating a large number of shuffles and capturing the output in a dataset, so some frequency distributions could be reviewed. In a large number of shuffles, the frequency distribution of the cards in any position of the hand should be 1/52 (about 1.9%), the frequency of any suit should be 1/4 (about 25%), and the frequency of any rank should be 1/13 (about 7.7%). The actual game play was tested by capturing the hand in a data set at each step of play for a reasonable number of games.

RESULTS

The table of frequencies and the bar chart in figure 2 shows the distribution of the number of cards left in the hand when 1 million trials of the game are simulated. The results show that ending up with no cards in the hand occurs less than 1% of the time, and ending up with a high number of cards (like 48 or more) is very rare.

CONCLUSION

Arrays used in a DATA step can be used to represent a standard deck of playing cards and can be used to simulate repeated trials of a card game in order to gain an understanding of the frequency distribution and probabilities of the possible outcomes of the game. The results of the simulation captured in a data set can then be analyzed using other SAS procedures.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Robert Moore
Ameriprise Financial
Ameriprise Financial Center
Minneapolis, MN 55474
Work Phone: 612-671-0858
E-mail: robert.2.moore@ampf.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

```
0001 %let num_games = 10000;                          /* the number of games to play */
0002 %let deck_size = 52;                         /* the number of cards in the deck */
0003
0004 data gamedata;
0005 array card{&deck_size} $ card1-card&deck_size;            /* an array of cards */
0006 format card1-card&deck_size $CHAR2.;
0007
0008 i = 1;                                            /* initialize deck of cards */
0009 do suit = 'H','C','D','S';
0010    do rank = '2','3','4','5','6','7','8','9','T','J','Q','K','A';
0011       card{i} = rank||suit;
0012       i=i+1;
0013       end;
0014    end;
0015 run;
0016
0017 data games;
0018 set gamedata;
0019 array card{&deck_size} $ card1-card&deck_size;            /* an array of cards */
0020 format card1-card&deck_size $CHAR2.;
0021 array hand{&deck_size} $ hand1-hand&deck_size;        /* an array for the hand */
0022 format hand1-hand&deck_size $CHAR2.;
0023
0024 do game_number=1 to &num_games;                        /* play multiple games */
0025
0026    do i=1 to &deck_size;                           /* shuffle the deck of cards */
0027       random_position = int(&deck_size * ranuni(0) + 1);      /* a random card */
0028       tempcard = card{i};                                  /* swap the ith card */
0029       card{i} = card{random_position};
0030       card{random_position} = tempcard;
0031       end;
0032
0033    t = 0;                        /* position of the rightmost card in the hand */
0034    do card_number=1 to &deck_size;
0035       t = t + 1;
0036       hand{t} = card{card_number};
0037       playing = 1;
0038       do while (playing and (t ge 4));     /* continue playing if have 4 cards */
0039          if (substr(hand{t},2,1) eq substr(hand{t-3},2,1))            /* suits */
0040              then do;                     /* discard the intervening two cards */
0041                   hand{t-2} = hand{t};
0042                   t = t - 2;
0043                   end;
0044              else do;
0045                   if (substr(hand{t},1,1) eq substr(hand{t-3},1,1))   /* ranks */
0046                       then t = t - 4;                  /* discard the 4 cards */
0047                       else playing = 0;     /* no suit or rank matches so stop */
0048                   end;
0049          end; /* while loop */
0050       end;
0051    cards_left = t;           /* the position of the rightmost card in the hand */
0052    output;       /* output the number of cards left in the hand (after 1 game) */
0053 end;
0054
0055 keep game_number cards_left;
0056 run;
0057
0058 proc freq data=games;
0059    tables cards_left;
0060 run;
```

Figure 1 - Program Code.

CARDS_LEFT	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	7,057	0.71	7,057	0.71
2	46,144	4.61	53,201	5.32
4	80,382	8.04	133,583	13.36
6	93,140	9.31	226,723	22.67
8	98,116	9.81	324,839	32.48
10	99,805	9.98	424,644	42.46
12	98,303	9.83	522,947	52.29
14	93,184	9.32	616,131	61.61
16	85,374	8.54	701,505	70.15
18	74,638	7.46	776,143	77.61
20	62,479	6.25	838,622	83.86
22	49,762	4.98	888,384	88.84
24	37,933	3.79	926,317	92.63
26	27,603	2.76	953,920	95.39
28	18,783	1.88	972,703	97.27
30	12,023	1.20	984,726	98.47
32	7,305	0.73	992,031	99.20
34	4,110	0.41	996,141	99.61
36	2,225	0.22	998,366	99.84
38	968	0.10	999,334	99.93
40	429	0.04	999,763	99.98
42	167	0.02	999,930	99.99
44	53	0.01	999,983	100.00
46	12	0.00	999,995	100.00
48	3	0.00	999,998	100.00
50	1	0.00	999,999	100.00
52	1	0.00	1,000,000	100.00

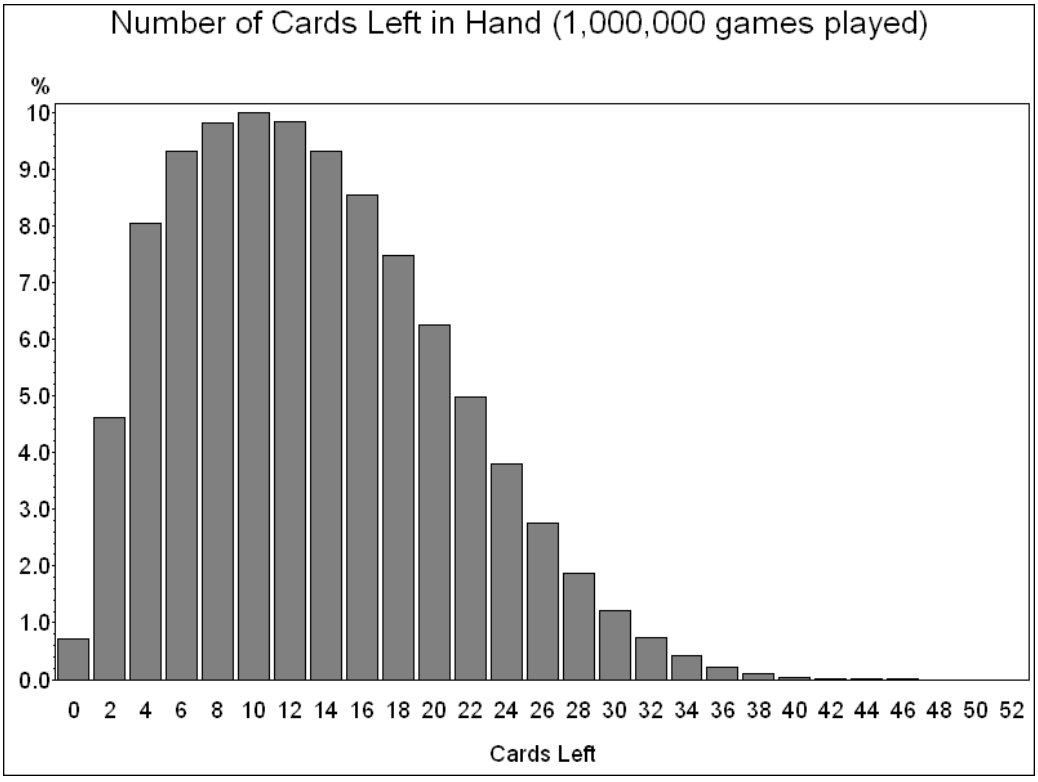


Figure 2 – Frequency distribution of the number of cards left in hand after 1 millions trials.