International Journal of Computer Applications (0975 – 8887)
Volume 54– No.11, September 2012

A Simulated Enhancement of Fisher-Yates Algorithm for Shuffling in Virtual Card Games using Domain-Specific Data Structures

Ade-Ibijola, Abejide Olu
Department of Computer Science,
Federal University of Technology, Akure, Nigeria.

ABSTRACT

Generally stated Algorithms are often preferred because of their wider horizon of applications owing to their freedom from any underlying language specification – they are not tied to any language and are easily implemented by anybody on any machine. This advantage becomes a hitch in applications where optimal implementation is of paramount concern – such as in games, as the developer's chosen language might suggest a better implementation that alters the algorithm using predefined data structures and/or sometimes, libraries of the preferred language. In this paper, a reconsideration of the Fisher-Yates Shuffle Algorithm was carried out using the "Generic List" data structure of the .Net library, an enhanced version of the Algorithm was drawn that targets a .Net implementation, a graphical simulation was developed using a "built-from-scratch" deck of Virtual Cards for the game of *Whot!* and a satisfactory permutation was achieved.

Keywords: Fisher-Yates Shuffle (FYS), Random Permutation, Domain-Specific Data Structures, Simulated Enhancement.

1. INTRODUCTION

Shuffling is a method employed to randomize a deck of playing cards to ensure an element of chance in card games. Literature has established that the computers are capable of generating an "ideal shuffle", a bias-free random permutation of cards. The Fisher-Yates shuffle (uncovered by Donald Knuth) is a trivial (seamless implementation) and pretty quick (with a complexity of O(n) on an n-card deck) algorithm for performing this [10]. However, a number of sub-standard algorithms are being widely used. A pretty trivial method is to assign random numbers to item in the deck, and then carry out sorting in the order of their respective random numbers. Even though this method has generated an acceptably random permutation, it is most suitable when none of the random numbers generated are repetitive.

However, there are crude ways of eliminating replicated random numbers. One major way is to adjust one of the repeated values randomly up or down by a little deviation, or tweak the probability of replication to a lower decimal by choosing an adequately wide range of random numbers. This method, if using an efficient sorting algorithm, will amount to an $O(n \log n)$ average and worst-case algorithm. Other frequently used algorithms have attempted to mimic manual shuffling with unacceptable performances: the permutations produced were highly predictable and they are slow.

FYS is regarded by many as an unbiased and optimal method for generating a truly random permutation of a finite set [10]. The idea of the Algorithm is relatively simple and similar to drawing-out numbered balls from a basket without replacement or blindly picking playing cards from a deck. Doing this computationally is the problem addressed by this Algorithm. An illustrative description of the FYS algorithm was given in [5]. Several implementations and variations of FYS exist. The original FYS was proposed in 1938 and reviewed in 1948 [5] with a modern version presented in [6]. A very similar but complete variant of the algorithm was published by Wilson in 2004 named the "Santtolo's Algorithm" [11]. In validating the FYS, a statistical analysis of the algorithm using frequency analysis was presented in [3] and conclusions from the analysis favoured the speed of this algorithm.

Irrespective of the unbiased nature of FYS, research reveals that there are often potential sources of bias such as: implementation errors, modulo bias and pseudorandom number generators not well-seeded [8]. This paper implements FYS in a new style, using the Generic List Object data structure on .Net Framework and adapts the algorithm to shuffle the deck of cards in the game of *Whot!* with an interactive computer simulation. *Whot*! is a card game whose description and rules of play is presented in [1].

The remaining of this paper is organized as follows: section 2 states the problem, section 3 reviews related work, section 4 briefly describes the original FYS with a flowchart, section 5 presents the enhanced version using the Generic List object that is specific to .Net framework, section 6 presents the results from the pseudo-permutation and screenshots from the simulation, and section 7 presents a concise conclusion and a peep into future work, succeeded with a list of references.

2. THE PROBLEM

The FYS is an established algorithm, tested and trusted, and as such, this work does not attempt to modify it in any way; instead, the challenge here is to implement FYS with the Generic List data structure (a luxury offered by the .Net Framework). The Generic List object data structure offers a higher abstraction compared to a mere Array. The implementation carried out and described in this work was adapted to the game of *Whot!*, hereby completing a first stage in the development of a virtual game of *whot!*

3. RELATED WORK

It is amazing how widely applied shuffling algorithms are; several variations of different algorithms and millions of implementations to match. To start with, Chen in 2012 used a model based on Cards-Shuffling to predict Urban Development in [9], implying that the kind of randomness generated in the process also finds applications in the society. Chen presented a schematic diagram of symmetry breaking and reconstruction of network of cities, having cards shuffling as a major component of his design as shown in Figure 1.

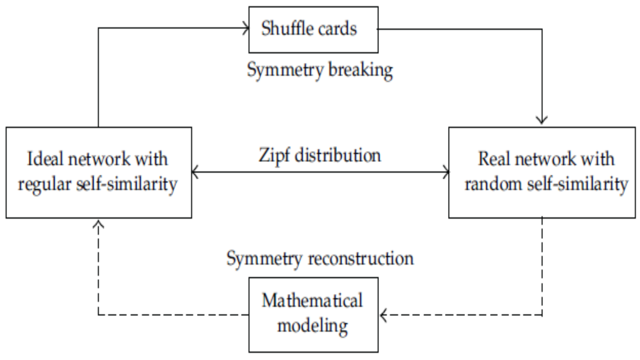

Fig. 1: Shuffling in Modeling of Urban Prediction (Adapted from [9])

Similarly, several implementations of different shuffling algorithms exist. An algorithm for shuffling an array was described in [4]. A naïve algorithm for shuffling was presented in [8] that randomizes by swapping positions of cards in a deck. This algorithm is shown in “Algorithm 1” below:

Algorithm 1: Naïve shuffle (Adapted from [8])

```
for (int i = 0; i < cards.Length; i++)
{
  int n = rand.Next(cards.Length);
  Swap(ref cards[i], ref cards[n]);
}
```

The method depicted by Algorithm 1 performs the following two steps:

(a). Iterates through each card in the deck.

(b). Swaps the current card’s position with another randomly chosen card.

The advantages of this algorithm are: simplicity, straightforwardness, ease of implementation (having few lines of code), and seemingly correct output. However, this was regarded by the author himself as a naïve algorithm, stating that it is suboptimal and consumes a large amount of resources (time, space, and memory access).

However, more complex versions of Algorithm 1 exist as variations of FYS. Furthermore, the question of “How many times should a deck of card be shuffled before randomness can be achieved?” was answered with proven illustrations in [2].

4. THE ORIGINAL FYS

The original FYS was presented in 1938 by Fisher and Yates [5] in the pages of their text “Statistical tables for biological, agricultural and medical research”. This original version was implemented manually, using pencil and paper, with a pre-determined table of random numbers to create an element of chance. A flowchart describing the steps of the original FYS is shown in Figure 2.

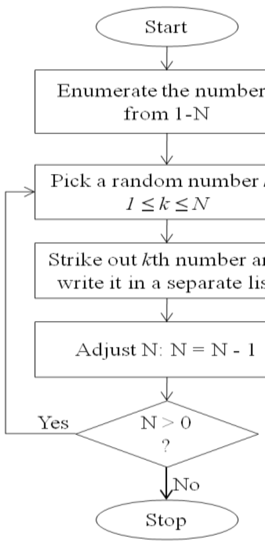
Start
Enumerate the numbers from 1-N
Pick a random number k; $1 \le k \le N$
Strike out kth number and write it in a separate list
Adjust N: N = N - 1
N > 0 ?
Yes
No
Stop

Fig. 2: Flowchart illustrating the steps of Original FYS

Given that the random numbers used in this method are unbiased, the process will definitely generate truly random permutations. Fisher and Yates also devised a way of obtaining random numbers to be used by the algorithm using predefined tables of random numbers [5].

5. ENHANCED FYS USING GENERIC LISTS

In this section we present a custom variation of FYS, written in Visual Basic.Net and targeted for the game of *Whot*! The pieces in the deck of the game of *Whot!* are characterized by both shapes and numbers which may be translated into a “Class” in VB.Net; therefore we present a code fragment that defines a VB Class (in Algorithm 2) as follows:

Algorithm 2: Cards Class in Vb.Net

```
Public Class cards
    'attriutes of the cards class ----
    Public symbol As String
    'circle, carpet, cross
    'triangle, star & king
    Public number As Integer 'from 1to14

    'methods here -----
End Class
```

The attributes of the class in “Algorithm 2” are symbol and number, while the possible values of any instance of the class “cards” is a 2-turple defined as:

```
[Symbol, Number]
```

where:

```
Symbol =
{circle,carpet,cross,triangle,star,king}

Number = {1..14}
```

We proceed to express our enhanced FYS algorithm using the Generic Lists of the class of “Cards” in “Algorithm 2” as follows:

Algorithm 3: Enhanced FYS with Generic Lists

```
Function shuffle_cards(ByVal cards As
     Generic.List(Of cards)) As
     Generic.List(Of cards)
      Dim rtn_cards As New Generic.List(Of
     cards)
      Dim n As Integer = cards.Count
      Dim rnd As New System.Random
      Dim pos As Integer = 1
      Do
        pos = rnd.Next(1, n + 1)
        rtn_cards.Add(cards(pos - 1))
        cards.RemoveAt(pos - 1)
        n -= 1
      Loop Until n <= 0
  Return rtn_cards
End Function
```

In Algorithm 3, we have used the “Cards” Class defined in Algorithm 2 as the Type of items contained in the Generic Lists. Hence, every item on the Generic List inherits the attributes of the “Card” Class, which are “Symbols and Numbers” as shown in Algorithm 2. Furthermore, we have used the methods of the Generic Lists such as `Add()` and `RemoteAt()` to “push to” and “pop from” the List. The complexity of Algorithm 3 is same has that of FYS, *O*(n) which is acceptable.

6. RESULTS & DISCUSSION

The Enhanced FYS applied to the game of *Whot!* (having 54 cards) in this paper produces exactly 54! (fifty-four factorial) possible orderings of the cards in a 54-card deck. This is a large number that is about:

```
230,843,697,339,241,379,243,718,839,060,267,085
,502,544,784,965,628,964,557,765,331,531,071,48
8.
```

Approximately, 2.3×10^{71} possible orderings.

The size of this number makes it almost impossible for two different permutations to be the same. However, it is possible to make some exact predictions based on probability, especially when a deck is not well randomized. This allows some degree of cheating in the game world.

6.1 Sample Permutations

Table 1 shows the permutations obtained from a deck of 20 cards and by mere inspection, we were able to sufficiently ascertain its randomness.

Table 1: 10 Permutations of a deck of 20 cards

		PERMUTATIONS									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
CARD POSITIONS	**1**	11	13	3	19	12	18	9	11	6	16
	2	3	6	13	7	7	9	19	2	20	17
	3	18	8	12	3	19	2	7	12	14	4
	4	12	7	17	1	13	4	20	14	19	6
	5	15	12	7	5	18	20	18	7	8	7
	6	5	17	11	17	5	14	8	19	15	2
	7	10	14	18	11	20	6	6	4	2	9
	8	4	18	6	15	14	1	10	9	16	19
	9	8	1	19	9	11	12	1	10	5	13
	10	6	19	14	12	2	15	13	13	4	8
	11	14	16	8	18	9	19	4	17	11	20
	12	7	15	15	6	15	5	5	5	17	5
	13	16	10	5	8	3	17	14	20	7	11
	14	13	11	10	13	6	3	12	3	12	15
	15	19	9	16	16	8	10	16	18	9	14
	16	9	3	9	14	4	11	11	15	18	18
	17	17	2	4	20	10	7	15	8	10	12
	18	1	4	1	10	1	16	3	1	3	1
	19	20	20	2	2	17	13	2	16	1	10
	20	2	5	20	4	16	8	17	6	13	3

6.2 Graphical Simulation

As earlier stated, we have adapted the Enhanced FYS to the game of *Whot!* as shown in Algorithm 3. However, since this implementation is a move towards developing the game itself, we deemed a graphical simulation necessary at this stage. Figure 3 shows 6 permutations of 10 out of 54 different cards of the *Whot!* game.

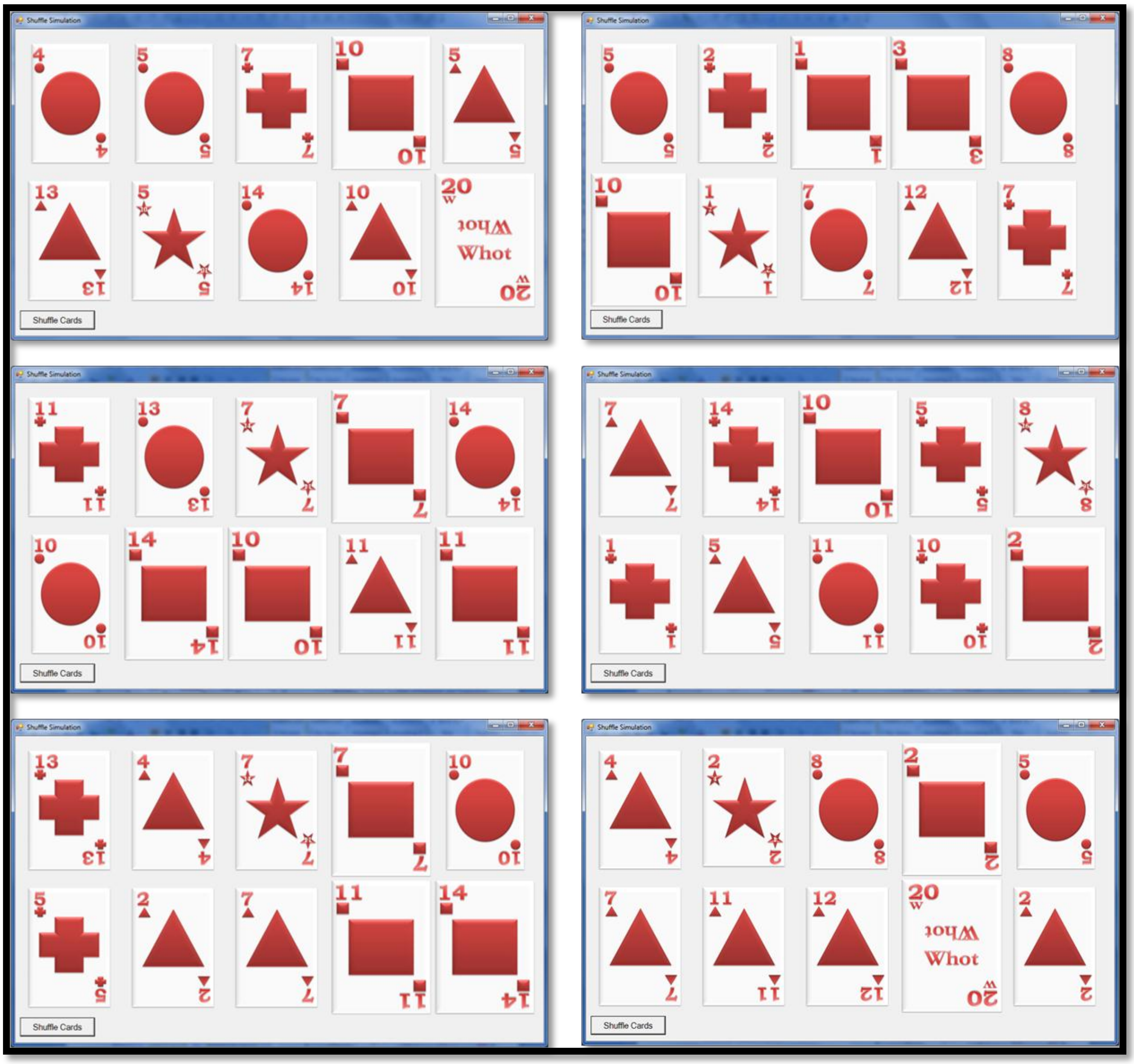

Fig. 3: Simulator Showing Six Distinct Permutations of *Whot!* cards

7. CONCLUSIONS & FUTURE WORK

In this work, we have considered the original FYS, enhanced it with a .Net data structure (Generic Lists), implemented it in a windows application using proprietary designs of virtual cards and obtained a satisfactory permutation, same complexity and speed. The enhanced algorithm described in this paper may be adopted by any interested individual to shuffle a deck of cards (or any array of finite items) on the described platform (.Net framework) or any other language library that has a similar object Data Structure with similar attributes and methods.

This work is, however, considered by the author as a first step towards building a virtual card game of *Whot!* and a prettier challenge envisioned and pinned-down for further investigation is how we may design a cognitive mental model for the Card Game robot (non-player character) that uses Dynamic Difficulty Balancing (DDB) to provide some reasonable level of challenge for a player across the playing-complexity curve.

REFERENCES

[1]. *McLeod John* (2007). Whot! Retrieved from http://www.pagat.com/com/whot.html#variations August, 2012.

[2]. Mann Brad (1993). How many Times Should you Shuffle a Deck of Cards? Dartmouth College Chance Project http://www.dartmouth.edu/~chance/teaching_aids/Mann.pdf

[3]. Mvngu (2011). Statistical Analysis of the Fisher-Yates Shuffle.http://mvngu.wordpress.com/2011/05/08/statistical-analysis-of-the-fisher-yates-shuffle/

[4]. Irde (2012). Shuffling an Array. Tutorial 2, inside Data Structures and Algorithms. http://www.lrde.epita.fr/~adl/ens/iitj/eso211/tut2.pdf

[5]. Fisher, R.A., Yates, F. (1948) [1938]. *Statistical tables for biological, agricultural and medical research* (3rd ed.). London: Oliver & Boyd. pp. 26–27. OCLC 14222135. (note: 6th edition, ISBN 0-02-844720-4, is available on the web, but gives a different shuffling algorithm by C. R. Rao)

[6]. Durstenfeld, Richard (1964). "Algorithm 235: Random permutation". *Communications of the ACM* **7** (7): 420. doi:10.1145/364520.364540.

[7]. Black, Paul E. (2005). "Fisher–Yates shuffle". *Dictionary of Algorithms and Data Structures*. National Institute of Standards and Technology.

[8]. Atwood Jeff (2007). Danger of Naivete. http://www.codinghorror.com/blog/2007/12/the-danger-of-naivete.html

[9]. Chen Yanguang (2012). Zipf's Law, Hierarchical Structure, and Cards-Shuffling Model for Urban Development. Hindawi Publishing Corporation. Discrete Dynamics in Nature and Society, Volume 2012, Article ID 480196, 21 pages.

[10]. Wiki (2012). Shuffling and Fisher-Yates Shuffle. Wikipedia Online Encyclopedia.

[11]. Wilson, M. C. (2004). "Overview of Sattolo's Algorithm"*INRIA Research Report*. 5542. Algorithms Seminar 2002–2004. Summary by Éric Fusy. pp. 105–108. ISSN 0249-6399.

.