**Introduction**

The debate around computerised random number generators and algorithms capable of generating supposedly random collections of numbers has been around since the early 1950s. Mathematician and computer scientist John Von Neumann was once quoted saying: *“Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin”* (**Von Neumann, 1951**). This is because arithmetical operations for creating random numbers, such as Von N’s own middle-square method, are not capable of generating a true random combination of numbers, instead producing pseudorandom sequences. Yet despite this lack of true random generation, pseudorandom algorithms are commonplace in all areas of computing, offering convenience in place of unreproducible data. Pseudorandom functions are provided as standard with almost all programming languages including C, C#, Python and JavaScript, with most allowing the user to define the seed used by the algorithm to allow for reproducible sequences. This notion of reproducible ‘random’ sequences brings a variety of challenges for both researchers and consumers in regard to areas such as cybersecurity or scientific testing where improper sequences could lead to faulty data or unsuitable cryptography. In addition, the validity of pseudorandom generators in digital environments that for instance shuffle or deal cards in games such as poker may not be an issue to a casual player, however for players who are competing for large pots of real-world money or in digital tournaments, ensuring that the system is fair and cannot be altered or predicted is essential.

A core issue faced historically by those looking to analyse random number generators can be explained by Park and Miller: strange and unpredictable is not necessarily random (**Park, Miller, 1988**). Even the most predictable pseudorandom generators, such as IBM’s notorious RANDU, are often capable of creating a sequence that at a glance appears to be random. Attempting to define what a random sequence looks like using only human intuition is an impossible task. Everyone has their own biases on what random looks like, from the expected number of sequential repetitions in a sequence to the ratio of heads to tails when flipping coins. The equanimity of your average tosser of coins depends upon a law…which ensures that he will not upset himself by losing too much nor upset his opponent by winning too often (**Stoppard, 1966**) and this extends to all areas of random data evaluation. It is only through visualisation and empirical testing that faults in these generators can be identified. Unlike with other forms of data where expected patterns allow for easy identification of outliers, random data must be viewed differently, with outlying data being assessed not on its correlation with other data points but rather on the individual probability of such an output occurring. It is for this reason that measures such as Chi-Squared Testing and Kolmogorov-Smirnov (KS) Testing are used. Tests such as these work by identifying the probability that a sequence contains the distribution of values seen. Results that are considered by these tests to be too likely or too unlikely are considered to be unsuitable. Other tests elaborate on this by providing a P-Value between 0 and 1 or a designated test statistic such as in the Anderson-Darling Test which is compared against alpha (α) to determine probability. The more unreliable a generator, the more tests it is likely to fail or otherwise perform poorly in. However, if a generator passes all tests, it does not mean that it is random since the right test matching the hidden pattern of the generated sequences may not have been found (**Luengo, et. Al, 2022**).

This project aimed to evaluate a collection of pseudorandom algorithms, to compare them to true random number generators and to test their versatility, seeing how close a pseudorandom algorithm is capable of getting to a true random generator. These tests addressed a variety of algorithms, including methods such as the middle-square, and compared them to physical random number generators, such as a shuffled deck of cards, dice and coins. The algorithms chosen for data collection all represented functions from in-use languages and sites as well as industry standard packages such as NumPy for Python. Since the applications of random number generation went beyond pure integer or decimal sequences, physical generators had to also be considered for this investigation. Decks of cards, dice and coins are frequently simulated by pseudorandom functions in a range of digital simulations and considering the requirement of their real-world counterparts to be unbiased and fair, including testing for them in both an empirical suite and simulated environment was considered essential. This can be highlighted in the analysis of the shuffled card decks, which were tested in a standard poker and a ‘Texas Hold Em’ style distribution. As well as comparing against true random generators, a collection of empirical tests originally described by Donald Knuth in his 1998 book: *The Art of Computer Programming Volume 2: Semi-numerical Algorithms* (**Knuth, 1998**) was used. These tests include the previously mentioned Chi-Squared and KS tests in addition to others such as the Birthday Spacings and Gap Test. An algorithm that ranked highly in these tests will be as ‘true to life’ as possible and be capable of generating a sequence that comes as close as possible to true random. The purpose of utilising an array of tests comes back to the idea of fault identification. Each test focusses on a different pattern of distribution and no generators were expected to be capable of passing all the tests provided. Where possible test results were grouped together for evaluation, primarily when comparing P-Values scored by the various generators across multiple tests.