Quantum Random Number Generator with Noise Analysis and Fairness Checks

My submission for Hackathon 2025 is a Quantum Random Number Generator (QRNG) constructed with Qiskit, with an emphasis on producing truly quantum random numbers and examining the effects of noise on fairness and randomness.

Typically, classical random number generators are pseudo-random. Since they employ deterministic algorithms, if the internal state or seed is known, their outputs should theoretically be predictable. On the other hand, this project creates essentially unpredictable results through quantum superposition and measurement. Without the need for an additional classical.random() step, each circuit run prepares qubits in a superposition using Hadamard (H) gates, measures them, and uses the measured bitstring as the random output. This guarantees that the number is not merely classically chosen from simulated data, but is actually quantumly random.

The QRNG is crafted to be adaptable, allowing for various numbers of qubits and outcomes. For instance, with 3 qubits, the circuit can produce 8 different outcomes, ranging from 000 to 111. These bitstrings are then translated into integers, which can be utilized in cryptographic protocols, simulations, or even simple demo applications. But it doesn’t stop at just generating a single random number; the project also enables the circuit to run multiple times to gather statistics. With these counts, I dive into analyzing how closely the distribution aligns with uniformity, how noise models (simulated through Qiskit Aer) affect the distribution, and whether basic error-mitigation techniques can enhance fairness. As part of a learning-focused extension, I also investigate concepts like using Bell states and entangled qubits to generate randomness, comparing their output distribution to that of independent qubits in superposition. All in all, this project serves as both a functional QRNG and a mini-laboratory for exploring quantum randomness, noise, and fairness using Qiskit.

qiskit-true-qrng-sachin/

- README.md
- LICENSE
- requirements.txt

qrng

- init__.py
- circuit.py   # build quantum circuits here
- runner.py   # run circuits and return numbers
- analysis.py   # fairness & noise analysis (optional)

`circuit.py` (concept)

- Function: `build_qrng_circuit(num_qubits: int)`
    - Create `QuantumCircuit(num_qubits, num_qubits)`
    - Apply H to all qubits
    - Measure all qubits

`runner.py` (concept)

Function: `run_qrng(num_qubits=3, backend="aer_simulator")`

Use Qiskit Aer simulator

Execute with `shots=1` for a single random number

Convert measured bitstring (like "010") to integer (2) and return it

Another function: `run_qrng_stats(num_qubits=3, shots=1024)`

Return counts dictionary and simple stats

`analysis.py`

•Functions to calculate:

- Expected ideal probability (= 1 / 2^num_qubits)
- Difference between observed and ideal
- Simple fairness score.

qiskit-true-qrng-sachin

Quantum Random Number Generator (QRNG) with Fairness and Noise Analysis

The QRNG is designed to be flexible, accommodating different numbers of qubits and outcomes. For example, with 3 qubits, the circuit can generate 8 unique outcomes, ranging from 000 to 111. These bitstrings are then converted into integers, which can be used in cryptographic protocols, simulations, or even straightforward demo applications. But it doesn’t just stop at producing a single random number; the project also allows the circuit to run multiple times to collect statistics. With these counts, I delve into analyzing how closely the distribution matches uniformity, how noise models (simulated through Qiskit Aer) influence the distribution, and whether basic error-mitigation techniques can improve fairness. As part of a learning-oriented extension, I also explore concepts like using Bell states and entangled qubits to generate randomness, comparing their output distribution to that of independent qubits in superposition. Overall, this project acts as both a practical QRNG and a mini-laboratory for investigating quantum randomness, noise, and fairness using Qiskit.

## Hackathon Idea and Goals

Detect AI-generated content and transform it into something that feels more human with our AI Content Detector. Just paste your text, and you'll receive accurate, natural-sounding results in no time! Here’s the text we’re diving into: ## Hackathon Idea and Goals This project revolves around four key concepts:

1. True QRNG core - Create a quantum circuit where all computational basis states ideally have equal chances. - Utilize the raw measurement bitstrings as random results.
2. Configurable system size- Give users the option to choose how many qubits they want. - Use n qubits to generate integers within the range of [0, $2n^{n- 1}$]
3. Fairness and distribution checks- Execute the circuit multiple times. - Analyze how closely the observed distribution matches the ideal uniform distribution.
4. Noise and error investigation - Leverage Qiskit Aer noise models to understand how noise impacts randomness. - Compare the deal simulation with the noisy one and explore ideas for error mitigation.

## Core QRNG Concept

1. Begin with n qubits all set to the state \|0…0⟩.
2. Use a Hadamard gate H on each qubit to create a uniform superposition across all 2^n basis states.
3. Measure all qubits in the computational basis.
4. Transform the resulting bitstring (like 101) into an integer (for instance,
5. In single-shot mode , each time you run the circuit, you get one quantum random number.
6. In multi-shot mode, we gather a large sample of results and analyze: - The frequency of each bitstring - Whether the distribution appears uniform - How noise affects the distribution

## Project Structure

qiskit-true-qrng-sachin/
- README.md
- requirements.txt
- LICENSE

qrng/
- init__.py
- circuit.py
- runner.py
- analysis.py
- xamples/
-     demo_qrng.py

File	What it’s responsible for
`__init__.py`	Makes this folder a Python package so you can import it.
`circuit.py`	Builds the quantum circuit that creates superposition and measures qubits.
`runner.py`	Runs the circuit and gives you random numbers (single-shot or multi-shot).
`analysis.py`	Extra tools for checking fairness, distributions, and noise effects.

Features

- **True quantum randomness**

The random numbers are generated *directly* from quantum measurement results. There is **no classical `.random()`** step that chooses among outcomes.

- **Configurable qubits**
    - Choose `num_qubits` depending on how many bits of randomness you need.
    - `num_qubits = 3` → integers from 0 to 7.
- **Single-shot and multi-shot modes**
    - Single-shot: get one random integer per execution.
    - Multi-shot: get a distribution over all possible outcomes.
- **Fairness metrics**
    - Compute frequencies of bitstrings.
    - Compare with the ideal uniform distribution.
    - Calculate a simple chi-square–like score to quantify deviation.

1. Generate a single quantum random number

```
from qrng.runner import run_qrng


number = run_qrng(num_qubits=3)
print(Quantum random number:, number)
```

This returns an integer between `0` and `7` (inclusive), directly derived from quantum measurement.

Generate many samples and see the distribution

```
from qrng.runner import run_qrng_distribution


counts = run_qrng_distribution(num_qubits=3, shots=1024)
print(Measurement counts:, counts)
```

You can then analyse fairness:

```
from qrng.analysis import describe_counts, chi_square_uniform


describe_counts(counts)
score = chi_square_uniform(counts)
print(Chi-square score vs uniform:, score)
```

Compare ideal vs noisy simulation

```
from qrng.runner import run_qrng_distribution
from qrng.analysis import compare_ideal_vs_noisy

ideal_counts, noisy_counts = compare_ideal_vs_noisy(
    num_qubits=3,
    shots=2048,
    single_qubit_error=0.01,
    two_qubit_error=0.02)

print("Ideal:", ideal_counts)
print("Noisy:", noisy_counts)
```

This illustrates how noise makes the distribution slightly biased, and opens the door to basic error-mitigation discussions.