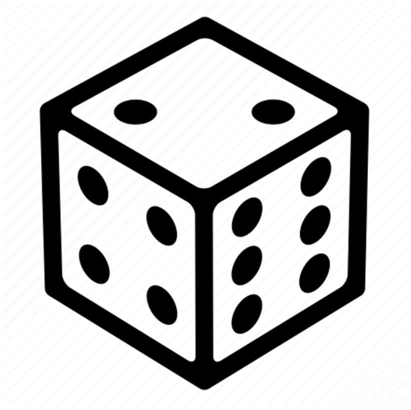
Generation of Quantum Randomness

Mentor: Marcel Pfaffhauser

Mentees: Edwin Agnew, Matthew Stypulkoski

Qiskit



Classical Randomness is **not** actually random

- Most classical randomness algorithms are pseudo-random
- The more random data needed, the worse the algorithms perform
    - **For Example**: Procedural Generation requires a lot of random data, but certain procedural generation tasks return quite poor results with classical randomness

- Utilizing the behavior of a qubit in superposition could result in more accurate, and actually random, randomness

Early Stages and Obstacles Faced

- We had goals set very early:
  - Allow basic users to use the project
  - Have the randomness returned as quickly as possible
  - Make sure the randomness was not pseudo-random
  - Be able to utilize our results in a Unity project

- Our largest obstacle was working with the speed at which results were returned.
  - The queue for the general use devices is very large
  - Real-time use of the method would be impossible

How to measure randomness?

- Shannon entropy measures the information content of a bitstring
  - If a coin has p = 1 of getting heads, flipping it tells you nothing => H = 0
  - If a coin has p = ½, you will be most “surprised” by every bit => H = 1
- However consider 1010101….
- 1-bit counts is random, 2-bit counts is constant

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- Use file compression algorithms instead?
  - Kolmogorov complexity

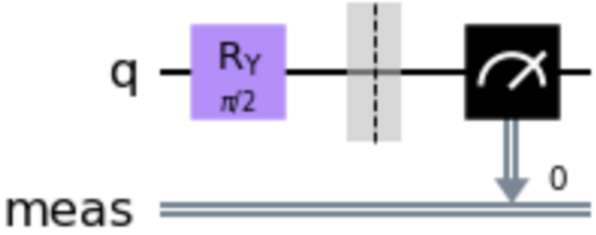
Basic Approach so far

NB Hadamard isn't purely random

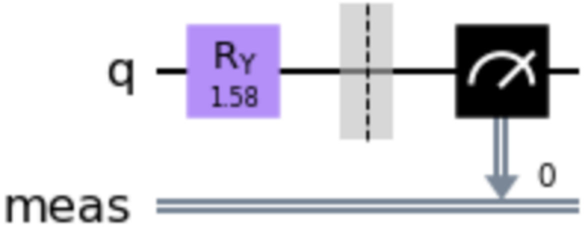
1. Initialize $\theta = \pi/2$
2. For $\theta_i \in \{(-k\delta) + \theta, \ldots, \theta, \ldots, (k\delta) + \theta\}$

   $R_y(\theta_i)$:

   a. Prepare
   b. Measure
   c. Run lots of times and compute entropy θ_i
3. Choose the which ma δ nizes the entropy and decrease



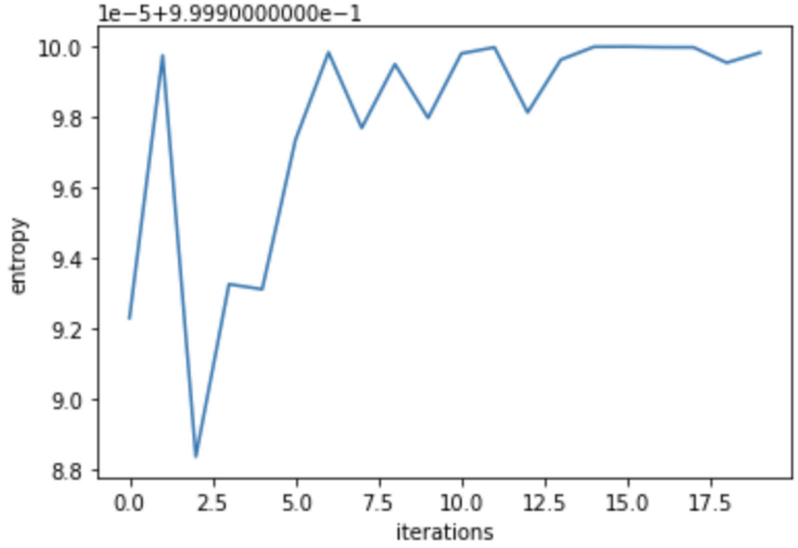

```
results: {'0': 4175, '1': 4017}
entropy = 0.9997316474300191
```



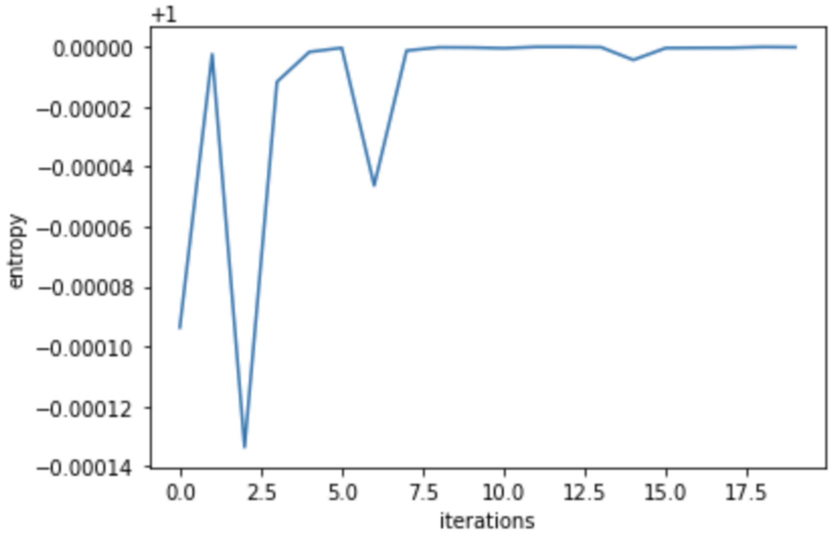

```
results: {'0': 4059, '1': 4133}
entropy = 0.9999411381372179
```

Results

- Simulated (with noise model)

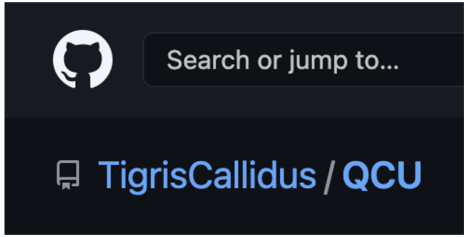


- Real device



Accessing Our Results in Unity

- We can use our Mentor's project, *QCU*, to run a python script in Unity




```python
from qiskit import(
  QuantumCircuit,
  execute,
  Aer)

simulator = Aer.get_backend('qasm_simulator')

qc = QuantumCircuit(5,5)
qc.h(0)
qc.h(1)
qc.h(2)
qc.h(3)
qc.h(4)
qc.measure([0,1,2,3,4], [0,1,2,3,4])

job = execute(qc, simulator, shots=1000)

result = job.result()
counts = result.get_counts(qc)
print(counts)
```

```
Counts:
{'00000': 36, '00001': 32, '10000': 29, '10001': 30, '10010': 32,
'10011': 26, '10100': 26, '10101': 31, '10110': 37, '10111': 34, '11000':
24, '11001': 34, '11010': 24, '11011': 36, '11100': 33, '11101': 28,
'11110': 40, '11111': 36, '00010': 33, '00011': 39, '00100': 36, '00101':
26, '00110': 42, '00111': 30, '01000': 28, '01001': 37, '01010': 33,
'01011': 25, '01100': 27, '01101': 29, '01110': 23, '01111': 24}
```

Where do we go from here...

- We can retrieve accurate randomness through Qiskit and IBM’s Quantum Computers
- We can access our circuit’s outputs in Unity

Now we need to work towards...

- Independent bits
- Expanding upon our circuit results in a Unity project
  - Convert the results into data useable by other Objects in the Unity project
  - Broaden the method to allow for more customizable parameters
- Packaging the logic into a Unity plugin
  - Allows for any user to use the quantum randomness in any Unity project

Questions?