Quantizing Procedural Generation with Qiskit

Henry Eigen, Vijay Rajagopal, Sai Thatigotla

Introduction

The area of quantum computing promises an exponential increase in computing power compared to current classical computing techniques. With players such as IBM, Google, and Microsoft heavily investing in creating reliable, usable quantum computers, there is interest in transforming classical algorithms with our current quantum computing capabilities. In this project, we attempt to interweave a quantum random number generator into an existing algorithm called “Wave Function Collapse” (WFC) in order to introduce true randomness in the algorithm.

Classical Algorithm Introduction

This section will discuss the general purpose and behavior of the aforementioned WFC algorithm. The original implementation of WFC was created by Maxim Gumin, a game developer, in 2016 and featured the ability to procedurally generate bitmaps with arbarity constraints (Mxgmn). Before getting into how the algorithm functions, one must understand the input and output space of WFC. WFC’s input space contains an x amount of “tile” inputs which will be used to satisfy an n unit space within a corresponding m dimensional space. For the sake of clarity, let’s use the original implementation’s example as reference; this would mean there would be an x amount of 10x10 pixel images that are going to be used to fill a 100x100 pixel image (with the total of 100 tiles to be filled).

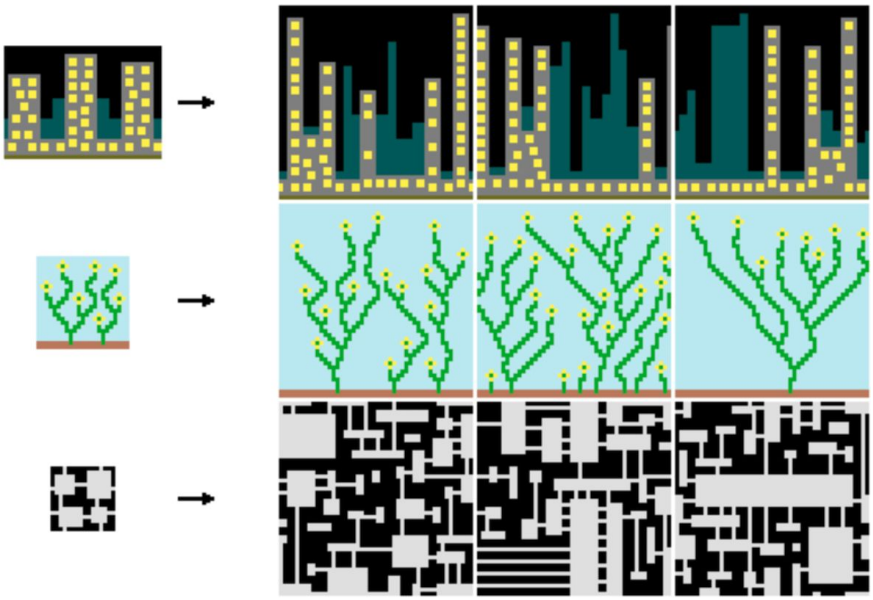


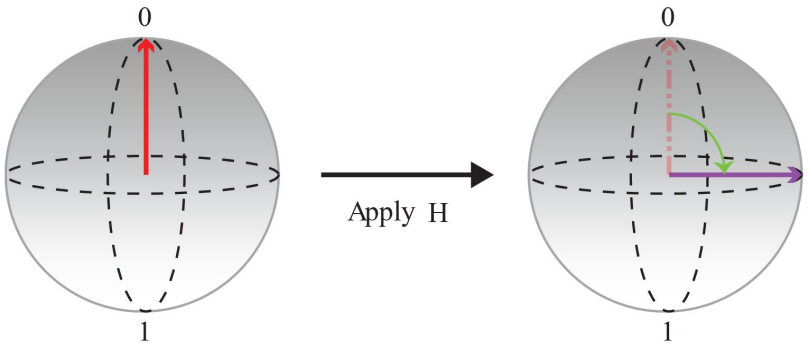
Figure 1. Examples of input & output from WFC (Mxgmn)

During the initialization of WFC, all unresolved tiles (which will be all 100 tiles) will have an equal probability of becoming all given inputs. This can be akin to all of the unresolved tiles being in a quantum-like “superposition” state. An input tile is initially randomly selected; for our example, let's assume WFC starts with the top left tile as the first space to “collapse” to a singular state. When the space collapses, this will cause a ripple throughout the rest of the probabilities within each of the unresolved tiles (Theaton). For the sake of brevity, let us call the action of collapsing a given space and recomputing the probabilities of the unresolved tiles as a “step”. The next tile to collapse will be determined by which tile has the lowest Shannon entropy of the unresolved tiles (Mxgmn). After choosing the tile with least Shannon entropy, WFC makes sure the current combination complies with a given rule set (Theaton). If the current combination and the input probabilities of unresolved tiles are in irresolvable conflict, then WFC will either reset **all** spaces and start over or, as in our implementation, revert an x number of collapsed tiles and recollapse them into new states. This aspect of the process can lead to extended runtimes depending on the frequency of the conflicts and the size of the output image.

Quantum Random Number Generator

The WFC algorithm is a classical algorithm that relies on a pseudo-random number generator (PRNG) to create the probability positions of each uninitialized tile during each step. PRNGs use certain formulas and tables to generate sequences of numbers that appear random. The main benefits of PRNGs are efficient and deterministic. The formulas are easy to compute for a computer and determinism helps with reproducibility if the seed used for the PRNG implementation is known. They can also be periodic, where the sequences will start repeating. However, the periods are large and usually are not a concern for most users. Overall, this makes PRNGs effective for most users and for some modeling applications where the replicability can be essential. However, if true randomness is needed, such as for encryption, PRNGs can be too predictable and true random-number generators are needed (TRNGs).

TRNGs work by observing natural phenomena to achieve randomness instead of using a computer (which is inherently deterministic). These phenomena vary from atmospheric noise to radioactive decay to perturbations in the movement of a computer mouse. TRNGs, unlike PRNGs, are inefficient, indeterministic, and nonperiodic. In this project, we use a Quantum Random Number generator (QRNG) which relies on the indeterminate and random nature of subatomic particles to extract randomness (Haahr). More specifically, we start with a qubit initialized to 0 and apply a Hadamard Gate to put into superposition and then measure to collapse the state into a 0 or 1. By doing this a *n* number of times, we can generate an *n*-bit string (Bromeg).



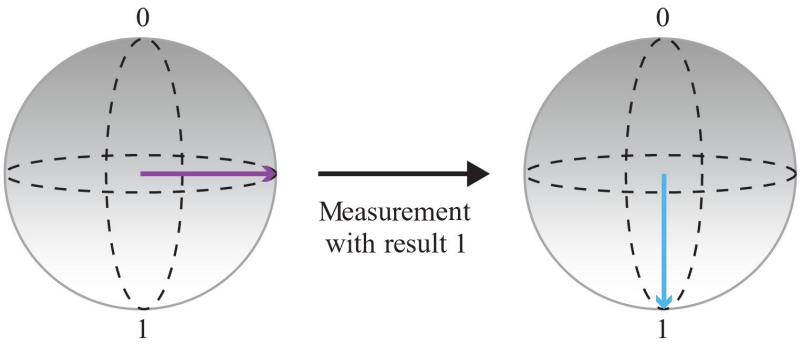


Figure 2. Bloch sphere representation of applying a Hadamard gate (Bromeg)

Our Implementation

Initially, our thought was to use Gumin's original implementation and create our planned QRNG in C#. Furthermore, our original modification plans were based on the fact that we could tie the uninitialized input tiles in superpositions and collapse all of them at once when one of them is measured. Unfortunately, with our time constraints, lack of expertise in C#, and more familiarity with IBM's Qiskit, we decided to use a ported Python version which purported to be using the same methodology as Gumin (https://github.com/yanfengliu/wfc). We decided to retrofit this existing implementation of WFC with quantum components. The most practical segment of WFC that could be swapped with a quantum circuit would be the PRNG, which was mentioned before as an integral part of generating unique states. In this section, we will discuss in detail the additions and modifications we did on Python WFC implementation.

The QRNG was constructed with IBM's Qiskit package for Python, which featured a five-qubit circuit with a corresponding Hadamard gate. Each of the qubits are measured and mapped to a single classical bit, which will carry the measured values of the 5 collapsed qubits and ultimately create a 5-bit binary number.

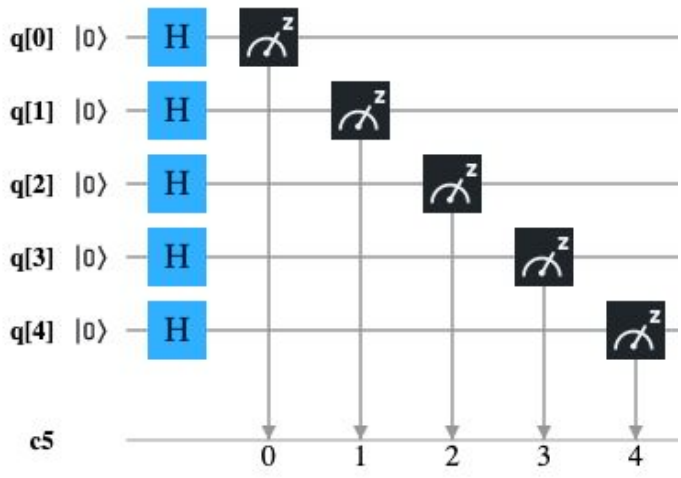
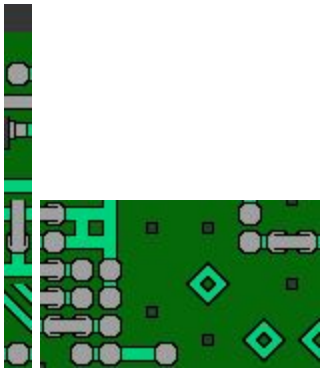


Figure 3. Circuit diagram of 5-qubit QRNG

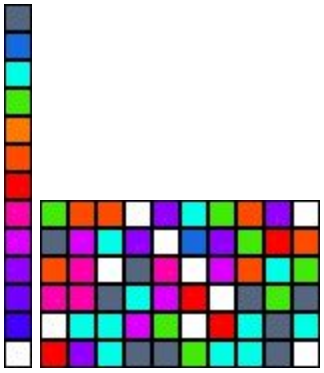
Additionally, we decided to add the ability to execute this quantum circuit on a real quantum computer by using IBM's "Quantum Experience". This service allows individuals the ability to queue and execute quantum circuits on IBM quantum computers around the world, but a circuit is limited in complexity and the number of qubits by the chosen quantum computer. The actual calling of the quantum computer is in utils/tiles.py within the comment block called "Calling Quantum Computer". Commenting out the code in that block and uncommenting the line below it, would use the simulator instead of IBM's actual quantum computers. The API key is placed in the API_token.txt file.

Apart from the QRNG, we also modified the image pipeline that was responsible for generating the input tiles used in WFC. The original implementation had a complex system of cropping the input images from a much larger image, which constrained the algorithm's flexibility with taking new patterns.

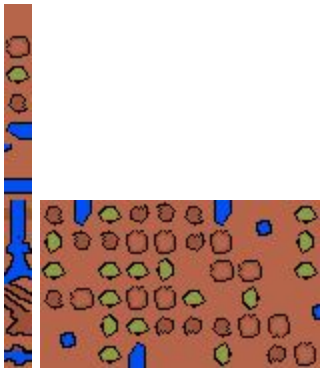
With all of these modifications, here are some of the inputs and outputs from our WFC implementation:



“Circuit” - Original Implementation Example



“Rainbow”



“Wild West”

Figure 4. 3 input tiles and generated images

Conclusion & Future Work

Although this project could not accomplish making a purely quantum-based implementation of WFC, we were able to add a quantum implementation of a RNG. This allows for pure randomness when a step needs to collapse to one of its available states. Unfortunately, WFC generates a random number numerous times. When calling actual quantum computers (in our case ibmq_vigo and ibmq_essex), it would take a significant amount of time (on the scale of hours when we tested with an actual quantum computer vs seconds with the simulator) for the jobs to be queued and executed. In a sense, the tradeoff in efficiency of using an actual quantum computer was too high to be effective in this case. The main future work would be to implement the entire algorithm with a quantum computer. Instead of its current approximation of superposition, an actual quantum implementation would map the states of the resulting image to a system of entangled qubits and collapse all the states simultaneously once one of the qubits is measured. The main drawback would be the amount of qubits required. Instead, an alternative algorithm or implementation might be needed to reduce the complexity of the circuit.

Works Cited

Bromeg. (n.d.). Create a Quantum Random Number Generator - Microsoft Quantum. Retrieved from https://docs.microsoft.com/en-us/quantum/quickstarts/qrng?tabs=tabid-qsharp

Haahr, M. (n.d.). Introduction to Randomness and Random Numbers. Retrieved from https://www.random.org/randomness/

Mxgmn. (2020, March 27). mxgmn/WaveFunctionCollapse. Retrieved from https://github.com/mxgmn/WaveFunctionCollapse

Theaton, R. (2018, December 17). The Wavefunction Collapse Algorithm explained very clearly. Retrieved from https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/