**Filename: interface.py**

**Object: abstract interfaces to represent the operations on a qubit and a quantum device**

1. Qubit 🡪 representation of fundamental operations on a qubit. It is used as main interface for concrete representations
   1. Method h()-> abstract method to represent the application of the Hadamard gate to the qubit
   2. Method measure(self)-> bool🡪 abstract method to measure the state of a qubit. It has to return a Boolean value, corresponding to the measurement made on the qubit
   3. Reset(self)-> abstract method that allows to reset the qubit to the |0> state.
2. QuantumDevice 🡪 abstract interface of a quantum device that allows for the processing of qubit, by allocating and deallocating it.
   1. Allocate\_qubit() 🡪 abstract method used for the allocation of the qubit
   2. Method deallocate\_qubit(self, qubit: Qubit) -> Qubit🡪 abstract method to deallocate the qubit
   3. Method using\_qubit(self) 🡪 it is a contenxt manager that automatically guarantees the processes of allocation, reset (even though there is an error) and deallocation.

Filename: simulator.py

Object: implements the concrete classes that are in grade of simulating the behavior of a qubit and single-qubit quantum device.

1. Constants

KET\_0 🡪 column vector to represent |0>

H 🡪 column vector to represent Hadamard gate matrix

1. Class SimulatedQubit

\_\_init\_\_(self) 🡪 initializes the qubit in the |0> state

h(self) 🡪 it applies the Hadamard transformation to the current state of the qubit

measure(self) -> bool 🡪 it computes the probability of measuring the state of the qubit |0> and returns a random Boolean value according to the probability. The self.state[0, 0] access to the first column and first row of the vector ([alfa], [beta]] where the square of alfa represents the probability of measuring 0.

pr0 = np.abs(self.state[0, 0]) \*\* 2 computes the theoretical probability of observing 0

sample = np.random.random() <= pr0 generates a random number within the interval (0.0, 1.0). if it is <= pr0 then it assumes a |0> state to be measured.

reset(self) 🡪 reset the state of the qubit to the |0>

1. Class SingleQubitSimulator

Available\_qubits: list containing the available qubits

Allocate\_qubit(self) -> SimulatedQubit 🡪 it returns a qubit from the list previously created

Deallocate\_qubit(self, qubit: SimulatedQubit) 🡪 it gives back a qubit to the list of the available ones.

Filename: qrng.py

Object: main module that uses the simulator to generate quantum random numbers

1. Function qnrg(device: QuantumDevice) -> bool:

It uses a simulated quantum device to generate a random single bit: allocates a qubit, then it applies a Hadamard transformation just to create a superposition that is measured in the end as a Boolean value.

1. Block if \_\_name\_\_ == “\_\_main\_\_” 🡪 it creates an instance SingleQubitSimulator and calls a hundred times qrng to randomly generate the number.

# Libraries needed

from abc import ABCMeta, ABC, abstractmethod

from contextlib import contextmanager

# Defines the basic functions a quantum computing program must have

class QuantumDevice(metaclass=ABCMeta):

possibility to allocate a single qubit for the user

@abstractmethod

  def allocate\_qubit(self) -> Qubit:

    pass

  Possibility to make the deallocation after using it

  @abstractmethod

  def deallocate\_qubit(self) -> Qubit:

    pass

We can define a context manager too simplify the process of allocation and deallocation of a qubit

Even if an exception is thrown up, the method allows to reset the qubit state and deallocate it immediately after.

  @contextmanager

  def using\_qubit(self):

    qubit = self.allocate\_qubit()

    try:

      yield qubit

    finally:

      qubit.reset()

      self.deallocate\_qubit()

**File: simulator.py**

**Content: implementation of a single qubit simulator**

**Elements**