Codes

quantumMedium.py

```
from flask import Flask, request
from qiskit import *

number_of_singlets = 500
qcomp = Aer.get_backend("qasm_simulator")
bases = {"alice": None, "bob": None, "eve": None}
results = None
singlets = []
chsh_res = None
chsh_corr_val = None

def alice_measure(qc, basis):  # basis = "X", "W" or "Z"
    """Alice's measurement circuits"""
    if basis == "X":  # X basis
        qc.h(0)
    elif basis == "W":  # W basis
        qc.s(0)
        qc.h(0)
        qc.t(0)
        qc.h(0)
    elif basis == "Z":  # Z basis
        pass
    else:
        raise ValueError("Value of parameter 'basis' can be one of 'X', 'W' or
'Z'")

def bob_measure(qc, basis):  # basis = "W", "Z" or "V"
    """Bob's measurement circuits"""
    if basis == "W":  # W basis
        qc.s(1)
        qc.h(1)
        qc.t(1)
        qc.h(1)
    elif basis == "Z":  # Z basis
        pass
    elif basis == "V":  # V basis
        qc.s(1)
        qc.h(1)
        qc.tdg(1)
        qc.h(1)
    else:
        raise ValueError("Value of parameter 'basis' can be one of 'W', 'Z' or
'V'")

def calc():
    global results, singlets, chsh_res
```

```
    alice_bases = bases["alice"]
    bob_bases = bases["bob"]
    eve_bases = bases["eve"]
    for i in range(number_of_singlets):
        singlet = QuantumCircuit(2, 4)
        singlet.x(0)
        singlet.x(1)
        singlet.h(0)
        singlet.cx(0, 1)
        if eve_bases:
            alice_measure(singlet, eve_bases[i])
            # bob_measure(singlet, eve_bases[i])
        alice_measure(singlet, alice_bases[i])
        bob_measure(singlet, bob_bases[i])
        singlet.measure(0, 0)
        singlet.measure(1, 1)
        singlets.append(singlet)
    result = execute(singlets, qcomp, shots=1).result().get_counts()
    chsh_res = result
    result = [list(x.keys())[0] for x in result]
    alice_result = "".join([x[-1] for x in result])
    bob_result = "".join([x[-2] for x in result])
    results = {"alice": alice_result, "bob": bob_result}

app = Flask(__name__)

@app.route("/")
def login():
    print(f"\nsomeone asked for the number of singlets... returned
{number_of_singlets}")
    return str(number_of_singlets)

@app.route("/send_bases", methods=["POST"])
def send_bases():
    global bases
    name = request.form["name"]
    bases[name] = request.form["bases"]
    print(f"\n{name.title()} sent choice of bases")
    if bases["bob"] and bases["alice"]:
        print("\nAs the Quantum Medium has both the bases of Alice and Bob, it can
start making measurements...")
        calc()
        print("\nmeasurements completed!")
    return "Ok"

@app.route("/measure_qubits", methods=["POST"])
def measure_qubits():
    name = request.form["name"]
    print(f"\n{name.title()} is requesting for the measurements.", end = " ")
    if results:
        print("Measurements are now ready, so Quantum Medium is  returning the
results.")
        return results[name]
    else:
```

```
        print("But measurements are not ready yet, so Quantum Medium requests
him/her to wait a bit before asking again!")
        return "wait"

app.run(
    host="localhost",
    port=5001,
    debug=True,
 )
```

classicalMedium.py

```
from flask import Flask, request

app = Flask(__name__)

bob_bases, alice_bases = None, None
bob_uBits, alice_uBits = None, None
status = None

@app.route('/get_bob_bases', methods=['POST'])
def get_bob_bases():
    global alice_bases
    alice_bases = request.form['bases']
    print(f"\nAlice sent her bases and is requesting for Bob's bases.", end = " ")
    if bob_bases:
        print("The Classical Medium already has Bob's bases, so it gives it to
her.")
        return bob_bases
    print("But Bob hasn't sent his bases yet, so she waits.")
    return "wait"

@app.route('/get_alice_bases', methods=['POST'])
def get_alice_bases():
    global bob_bases
    bob_bases = request.form['bases']
    print(f"\nBob sent his bases and is requesting for Alice's bases.", end = " ")
    if alice_bases:
        print("The Classical Medium already has Alice's bases, so it gives it to
him.")
        return alice_bases
    print("But Alice hasn't sent her bases yet, so he waits.")
    return "wait"

@app.route('/get_bob_uBits', methods=['POST'])
def get_bob_uBits():
    global alice_uBits
    alice_uBits = request.form['uBits']
    print(f"\nAlice sent her unused Bits and is requesting for Bob's unused
Bits.", end = " ")
```

```
    if bob_uBits:
        print("The Classical Medium already has Bob's unused Bits, so it gives it
to her.")
        return bob_uBits
    print("But Bob hasn't sent his unused Bits yet, so she waits.")
    return "wait"

@app.route('/get_alice_uBits', methods=['POST'])
def get_alice_uBits():
    global bob_uBits
    bob_uBits = request.form['uBits']
    print(f"\nBob sent his unused Bits and is requesting for Alice's unused
Bits.", end = " ")
    if alice_uBits:
        print("The Classical Medium already has Alice's unused Bits, so it gives
it to him.")
        return alice_uBits
    print("But Alice hasn't sent her unused Bits yet, so he waits.")
    return "wait"

app.run(
    host = "localhost",
    port = 5000,
    debug = True,
 )
```

module.py

```
import random
import requests
from time import sleep

qIP, qPORT = "localhost", "5001"
cIP, cPORT = "localhost", "5000"
quantuMedium  =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel


def send_bases(baseOpt, number_of_singlets, name):
    choice = "".join([baseOpt[random.randint(0, len(baseOpt)-1)] for i in
range(number_of_singlets)])
    print("Sending bases...")
    requests.post(f"{quantuMedium}send_bases",
                  data = {
                      "name": name,
                      "bases": choice,
                      }
                  ).text
    return choice

def measure_qubits(name):
```

```
    try:
        print("Reading qubits...")
        ret = requests.post(f"{quantuMedium}measure_qubits", data=
{"name":name}).text
    except:
        sleep(10)
        return measure_qubits(name)
    if  ret == "wait":
        sleep(10)
        return measure_qubits(name)
    return ret

def get_bob_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_bases", data = {"bases":
bases}).text
    except:
        sleep(1)
        return get_bob_bases(bases)
    if  ret == "wait":
        sleep(1)
        return get_bob_bases(bases)
    return ret

def get_alice_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_alice_bases", data = {"bases":
bases}).text
    except:
        sleep(1)
        return get_alice_bases(bases)
    if  ret == "wait":
        sleep(1)
        return get_alice_bases(bases)
    return ret

def get_bob_uBits(uBits):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_uBits", data = {"uBits":
uBits}).text
    except:
        sleep(1)
        return get_bob_uBits(uBits)
    if  ret == "wait":
        sleep(1)
        return get_bob_uBits(uBits)
    return ret

def get_alice_uBits(uBits):
    try:
        ret = requests.post(f"{classicalMedium}get_alice_uBits", data = {"uBits":
uBits}).text
    except:
        sleep(1)
```

```
        return get_alice_uBits(uBits)
    if  ret == "wait":
        sleep(1)
        return get_alice_uBits(uBits)
    return ret

def chsh(uBitsA, uBitsB, alice_bases, bob_bases):
    search = ["00", "01", "10", "11"]  # f"{bob}{alice}"
    XW = [0, 0, 0, 0]
    XV = [0, 0, 0, 0]
    ZW = [0, 0, 0, 0]
    ZV = [0, 0, 0, 0]

    for i in range(len(alice_bases)):
        try:
            if uBitsA[i] == " ":
                continue
        except IndexError:
            raise IndexError(f"i = {i}")

        if alice_bases[i] == "X" and bob_bases[i] == "W":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    XW[j] += 1
                    break
        if alice_bases[i] == "X" and bob_bases[i] == "V":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    XV[j] += 1
                    break
        if alice_bases[i] == "Z" and bob_bases[i] == "W":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    ZW[j] += 1
                    break
        if alice_bases[i] == "Z" and bob_bases[i] == "V":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    ZV[j] += 1
                    break

    # expectation values of XW, XV, ZW and ZV observables (2)
    chsh_corr_val = [
                (XW[0] - XW[1] - XW[2] + XW[3]) / sum(XW),  # -1/sqrt(2)
                -(XV[0] - XV[1] - XV[2] + XV[3]) / sum(XV),  # 1/sqrt(2)
                (ZW[0] - ZW[1] - ZW[2] + ZW[3]) / sum(ZW),  # -1/sqrt(2)
                (ZV[0] - ZV[1] - ZV[2] + ZV[3]) / sum(ZV)  # -1/sqrt(2)
        ]

    chsh_corr_val = sum(chsh_corr_val)
    return chsh_corr_val
```

alice.py

```
from module import *

name = "alice"
print(f"Hi! This is {name.title()}!")
baseOpt = ['X', 'W', 'Z']

number_of_singlets = int(requests.get(quantuMedium).text)
alice_bases = send_bases(baseOpt, number_of_singlets, name)
bits = measure_qubits(name)
bob_bases = get_bob_bases("".join(alice_bases))

key = ""
uBitsA = ""
for i in range(number_of_singlets):
    if alice_bases[i] == bob_bases[i]:
        bit = int(bits[i])
        if bit: key += "1"
        else: key += "0"
        uBitsA += " "
    else:
        uBitsA += bits[i]

key_length = len(key)
mismatch = number_of_singlets - key_length

uBitsB = get_bob_uBits(uBitsA)

chsh_score = float(chsh(uBitsA, uBitsB, alice_bases, bob_bases))
print(f"chsh score: {chsh_score}")
success = chsh_score < -2

if success:
    print(key)
else:
    print("Eve is here! Abort!")
```

bob.py

```
from module import *

name = "bob"
print(f"Hi! This is {name.title()}!")
baseOpt = ['W', 'Z', 'V']

number_of_singlets = int(requests.get(quantuMedium).text)
bob_bases = send_bases(baseOpt, number_of_singlets, name)
bits = measure_qubits(name)
alice_bases = get_alice_bases("".join(bob_bases))
```

```
key = ""
uBitsB = ""
for i in range(number_of_singlets):
    if bob_bases[i] == alice_bases[i]:
        bit = int(bits[i])
        if bit: key += "0"
        else: key += "1"
        uBitsB += " "
    else:
        uBitsB += bits[i]

key_length = len(key)
mismatch = number_of_singlets - key_length

uBitsA = get_alice_uBits(uBitsB)

chsh_score = chsh(uBitsA, uBitsB, alice_bases, bob_bases)
print(f"chsh score: {chsh_score}")
success = chsh_score < -2

if success:
    print(key)
else:
    print("Eve is here! Abort!")
```

eve.py

```
from module import *

name = "eve"
print(f"Hi! This is {name.title()}!")
baseOpt = ['W', 'Z']

singlets_sent = int(requests.get(quantuMedium).text)
send_bases(baseOpt, singlets_sent, name)
print("completed eavesdropping!")
```