In this exercise, we explore how one can cheat with a quantum computer in a classical well known game: The coin toss game. In this kind of games, two players take turns flipping a coin. The goal is to predict whether the coin will land on heads or tails. Assume that the computer plays first and then you play second and then the computer, to play fair, can play again. If the coin at the end is 0 the computer win and if is 1 then you win.

1-Provide a python program that simulates the game and conclude that the game is fair.

Regular Python code running cointoss and quantum version of it

```
import random

def coin_toss():
    return random.randint(0, 1)

def play_game():
    computer_bet = 0
    player_bet = 1

    # computer plays:
    coin_val = coin_toss()

    # player plays
    if coin_val == computer_bet:
        coin_val = coin_toss()

        #computer plays
        if coin_val == player_bet:
            coin_val = coin_toss()

    return coin_val

n = 0
for _ in range(1000):
    val = play_game()
    n += val

print("player won ", n, "times")
```

```
player won  662 times
```

```
from qiskit import *
from qiskit_aer import Aer
import numpy as np

def coin_toss():
    qbit = QuantumRegister(1, 'qubit')
    cbit = ClassicalRegister(1, 'classical')
```

```
    qc = QuantumCircuit(qbit,cbit)
    initial_state = 0
    qc.initialize(initial_state, 0)
    qc.h(0)
    res = qc.measure(0,0)
    simulator = Aer.get_backend('qasm_simulator')
    circ = transpile(qc, simulator)
    result = simulator.run(circ, shots=1).result()
    counts = result.get_counts(circ)
    if '0' in counts:
        return 0
    else:
        return 1

def play_game():
    computer_bet = 0
    player_bet = 1

    # computer plays:
    coin_val = coin_toss()

    # player plays
    if coin_val == computer_bet:
        coin_val = coin_toss()

        #computer plays
        if coin_val == player_bet:
            coin_val = coin_toss()


    return coin_val

n = 0
for _ in range(100):
    val = play_game()
    n += val

print("player won ", n, "times")
```

```
player won  57 times
```

2 - Now, think of a quantum strategy that would allow the computer to always win. Explain you thinking.

Answer:

For the regular coin toss we do use the Hadamard gate. It does put the qubit that was in the base state 0 in an equal superposition of states 1 and 0. So two interesting strategies would be:

- to to the reverse operation. Set the initial state to that superposition and then apply the hadamard gate.
- leave the initial state as 0 and apply Hadamard gate twice

for the first if the PC is picking 0 as a result for example we can set the initial state to [-1/np.sqrt(2), -1/np.sqrt(2)] and when measured this will always read 0. For the second is pretty much the same because that's what you are doing by using Hadamard gate twice after starting with 0, the end result will be the same.

3- Implement it in Qiskit.

```
from qiskit import *
from qiskit_aer import Aer
from qiskit.quantum_info import Statevector
import numpy as np

def coin_toss():
    qbit = QuantumRegister(1, 'qubit')
    cbit = ClassicalRegister(1, 'classical')
    qc = QuantumCircuit(qbit,cbit)
    initial_state = [-1/np.sqrt(2), -1/np.sqrt(2)]
    qc.initialize(initial_state, 0)
    qc.h(0)
    res = qc.measure(0,0)
    simulator = Aer.get_backend('qasm_simulator')
    circ = transpile(qc, simulator)
    result = simulator.run(circ, shots=1).result()
    counts = result.get_counts(circ)
    if '0' in counts:
        return 0
    else:
        return 1

def play_game():
    computer_bet = 0
    player_bet = 1

    # computer plays:
    coin_val = coin_toss()

    # player plays
    if coin_val == computer_bet:
        coin_val = coin_toss()

        #computer plays
        if coin_val == player_bet:
            coin_val = coin_toss()


    return coin_val
```

```
print("Setting initial state")
n = 0
for _ in range(100):
    val = play_game()
    n += val

print("player won ", n, "times")

## Simplified version:
print("simplified version")
qbit = QuantumRegister(1, 'qubit')
cbit = ClassicalRegister(1, 'classical')
qc = QuantumCircuit(qbit,cbit)
initial_state = Statevector([1/np.sqrt(2), 1/np.sqrt(2)])
print("state is valid ", initial_state.is_valid())
qc.initialize(initial_state, 0)
qc.h(0)
res = qc.measure(0,0)
simulator = Aer.get_backend('qasm_simulator')
circ = transpile(qc, simulator)
result = simulator.run(circ, shots=2048).result()
counts = result.get_counts(circ)
print(counts)

## Two Hadamard version:
print("simplified version")
qbit = QuantumRegister(1, 'qubit')
cbit = ClassicalRegister(1, 'classical')
qc = QuantumCircuit(qbit,cbit)
initial_state = 0
qc.initialize(initial_state, 0)
qc.h(0)
qc.h(0)
res = qc.measure(0,0)
simulator = Aer.get_backend('qasm_simulator')
circ = transpile(qc, simulator)
result = simulator.run(circ, shots=2048).result()
counts = result.get_counts(circ)
print(counts)
```

```
Setting initial state
player won  0 times
simplified version
state is valid  True
{'0': 2048}
simplified version
{'0': 2048}
```

4- What happens if you play with a biased coin?

Answer:

If one plays with a biased coin, say the example given of initial state: [-1/np.sqrt(2), -1/np.sqrt(2)] the result will always be the same, 0. The same is true for a state that would always convert to 1 such as [1/np.sqrt(2), -1/np.sqrt(2)]. Also there are initial state combinations where this won`t always be true, but extremly probable such as: [np.sqrt(3)/2,np.e**(1j*np.pi)/2]

```
## Biased coin being tossed
print("Regular game")
qbit = QuantumRegister(1, 'qubit')
cbit = ClassicalRegister(1, 'classical')
qc = QuantumCircuit(qbit,cbit)
initial_state = [np.sqrt(3)/2,np.e**(1j*np.pi)/2]
qc.initialize(initial_state, 0)
qc.h(0)
res = qc.measure(0,0)
simulator = Aer.get_backend('qasm_simulator')
circ = transpile(qc, simulator)
result = simulator.run(circ, shots=2048).result()
counts = result.get_counts(circ)
print(counts)
```

```
Regular game
{'0': 119, '1': 1929}
```

```
print("Double Hadamard strategy")
qbit = QuantumRegister(1, 'qubit')
cbit = ClassicalRegister(1, 'classical')
qc = QuantumCircuit(qbit,cbit)
initial_state = [np.sqrt(3)/2,np.e**(1j*np.pi)/2]
qc.initialize(initial_state, 0)
qc.h(0)
qc.h(0)
res = qc.measure(0,0)
simulator = Aer.get_backend('qasm_simulator')
circ = transpile(qc, simulator)
result = simulator.run(circ, shots=2048).result()
counts = result.get_counts(circ)
print(counts)
```

```
Double Hadamard strategy
{'1': 517, '0': 1531}
```

5- What happens if instead of starting with 0 you would start with a 1?

Answer:

If starting with 1 instead of zero the state of superposition would be: [1/np.sqrt(2), -1/np.sqrt(2)] which would leave each coin toss with the regular 50% chance, so the rest of the game would play out as normal. If we are using the computer always win strategy for a fixed initial state like 1 And we apply Hadamard gate twice then the result will always be 1 and player will always win