# T2 - Entanglement - CHSH Game - ParsaVARES

November 1, 2024

## 1 Entanglement in Action

## 2 CHSH Game

[1]:
```
%pip install qiskit[visualization]
```

```
Collecting qiskit[visualization]
  Using cached
qiskit-1.2.4-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(12 kB)
Collecting rustworkx>=0.15.0 (from qiskit[visualization])
  Using cached rustworkx-0.15.1-cp38-abi3-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.9 kB)
Requirement already satisfied: numpy<3,>=1.17 in /opt/conda/lib/python3.11/site-
packages (from qiskit[visualization]) (1.26.4)
Collecting scipy>=1.5 (from qiskit[visualization])
  Using cached
scipy-1.14.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(60 kB)
Collecting sympy>=1.3 (from qiskit[visualization])
  Using cached sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Collecting dill>=0.3 (from qiskit[visualization])
  Using cached dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: python-dateutil>=2.8.0 in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (2.9.0)
Collecting stevedore>=3.0.0 (from qiskit[visualization])
  Using cached stevedore-5.3.0-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (4.11.0)
Collecting symengine<0.14,>=0.11 (from qiskit[visualization])
  Using cached symengine-0.13.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: matplotlib>=3.3 in
/opt/conda/lib/python3.11/site-packages (from qiskit[visualization]) (3.9.2)
Collecting pydot (from qiskit[visualization])
  Using cached pydot-3.0.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: Pillow>=4.2.1 in /opt/conda/lib/python3.11/site-
packages (from qiskit[visualization]) (11.0.0)
```

```
Collecting pylatexenc>=1.4 (from qiskit[visualization])
  Using cached pylatexenc-2.10-py3-none-any.whl
Collecting seaborn>=0.9.0 (from qiskit[visualization])
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.11/site-
packages (from matplotlib>=3.3->qiskit[visualization]) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/conda/lib/python3.11/site-packages (from
matplotlib>=3.3->qiskit[visualization]) (3.2.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.8.0->qiskit[visualization]) (1.16.0)
Requirement already satisfied: pandas>=1.2 in /opt/conda/lib/python3.11/site-
packages (from seaborn>=0.9.0->qiskit[visualization]) (2.2.3)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit[visualization])
  Using cached pbr-6.1.0-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy>=1.3->qiskit[visualization])
  Using cached mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.11/site-
packages (from pandas>=1.2->seaborn>=0.9.0->qiskit[visualization]) (2024.2)
Using cached dill-0.3.9-py3-none-any.whl (119 kB)
Using cached
rustworkx-0.15.1-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0
MB)
Using cached
scipy-1.14.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (41.2
MB)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Using cached stevedore-5.3.0-py3-none-any.whl (49 kB)
Using cached
symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(49.7 MB)
Using cached sympy-1.13.3-py3-none-any.whl (6.2 MB)
Using cached pydot-3.0.2-py3-none-any.whl (35 kB)
Using cached
```

```
qiskit-1.2.4-cp38-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.8 MB)
Using cached mpmath-1.3.0-py3-none-any.whl (536 kB)
Using cached pbr-6.1.0-py2.py3-none-any.whl (108 kB)
Installing collected packages: pylatexenc, mpmath, sympy, symengine, scipy,
rustworkx, pydot, pbr, dill, stevedore, seaborn, qiskit
Successfully installed dill-0.3.9 mpmath-1.3.0 pbr-6.1.0 pydot-3.0.2
pylatexenc-2.10 qiskit-1.2.4 rustworkx-0.15.1 scipy-1.14.1 seaborn-0.13.2
stevedore-5.3.0 symengine-0.13.0 sympy-1.13.3
Note: you may need to restart the kernel to use updated packages.
```

[2]:

```
%pip install qiskit_aer
```

```
Collecting qiskit_aer
  Using cached qiskit_aer-0.15.1-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.0 kB)
Requirement already satisfied: qiskit>=1.1.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit_aer) (1.2.4)
Requirement already satisfied: numpy>=1.16.3 in /opt/conda/lib/python3.11/site-
packages (from qiskit_aer) (1.26.4)
Requirement already satisfied: scipy>=1.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit_aer) (1.14.1)
Requirement already satisfied: psutil>=5 in /opt/conda/lib/python3.11/site-
packages (from qiskit_aer) (5.9.8)
Requirement already satisfied: rustworkx>=0.15.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit>=1.1.0->qiskit_aer) (0.15.1)
Requirement already satisfied: sympy>=1.3 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit>=1.1.0->qiskit_aer) (1.13.3)
Requirement already satisfied: dill>=0.3 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit>=1.1.0->qiskit_aer) (0.3.9)
Requirement already satisfied: python-dateutil>=2.8.0 in
/opt/conda/lib/python3.11/site-packages (from qiskit>=1.1.0->qiskit_aer) (2.9.0)
Requirement already satisfied: stevedore>=3.0.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit>=1.1.0->qiskit_aer) (5.3.0)
Requirement already satisfied: typing-extensions in
/opt/conda/lib/python3.11/site-packages (from qiskit>=1.1.0->qiskit_aer)
(4.11.0)
Requirement already satisfied: symengine<0.14,>=0.11 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
qiskit>=1.1.0->qiskit_aer) (0.13.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.11/site-
packages (from python-dateutil>=2.8.0->qiskit>=1.1.0->qiskit_aer) (1.16.0)
Requirement already satisfied: pbr>=2.0.0 in
```

```
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
stevedore>=3.0.0->qiskit>=1.1.0->qiskit_aer) (6.1.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/opt/.qbraid/environments/qbraid_000000/pyenv/lib/python3.11/site-packages (from
sympy>=1.3->qiskit>=1.1.0->qiskit_aer) (1.3.0)
Using cached
qiskit_aer-0.15.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(12.3 MB)
Installing collected packages: qiskit_aer
Successfully installed qiskit_aer-0.15.1
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
# Required imports
from qiskit import QuantumCircuit
from qiskit_aer.primitives import Sampler
from numpy import pi
from numpy.random import randint
```

We can implement the CHSH game together with the quantum strategy defined above in Qiskit as follows.

First, here's the definition of the game itself, which allows an arbitrary strategy to be plugged in as an argument.

[5]:
```
def chsh_game(strategy):
    """Plays the CHSH game
    Args:
        strategy (callable): A function that takes two bits (as `int`s) and
            returns two bits (also as `int`s). The strategy must follow the
            rules of the CHSH game.
    Returns:
        int: 1 for a win, 0 for a loss.
    """
    # Referee chooses x and y randomly
    x, y = randint(0, 2), randint(0, 2)

    # Use strategy to choose a and b
    a, b = strategy(x, y)

    # Referee decides if Alice and Bob win or lose
    if (a != b) == (x & y):
        return 1  # Win
    return 0  # Lose
```

Now we'll create a function that outputs a circuit depending on the questions for Alice and Bob. We'll let the qubits have their default names for simplicity, and we'll use the built-in Ry( ) gate for Alice and Bob's actions.

[4]:

```
def chsh_circuit(x, y):
    """Creates a `QuantumCircuit` that implements the best CHSH strategy.
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        QuantumCircuit: Circuit that, when run, returns Alice and Bob's
            answer bits.
    """
    # Replace ?
    qc = QuantumCircuit(2, 2)
    # Initialize qubits in entangled state
    qc.h(0)             # Apply Hadamard gate to Alice's qubit (index 0)
    qc.cx(0, 1)         # Apply CNOT gate with Alice's qubit as control and Bob's qubit as target
    qc.barrier()
    qc.barrier()

    # Alice
    if x == 0:
        qc.ry(0, 0)  # No rotation if x is 0 (Ry(0) is the identity operation)
    else:
        qc.ry(pi / 2, 0)  # Rotate by  /2 if x is 1
    qc.measure(0, 0)

    # Bob
    if y == 0:
        qc.ry(pi / 4, 1)  # Rotate by  /4 if y is 0
    else:
        qc.ry(-pi / 4, 1)  # Rotate by - /4 if y is 1
    qc.measure(1, 1)

    return qc
```

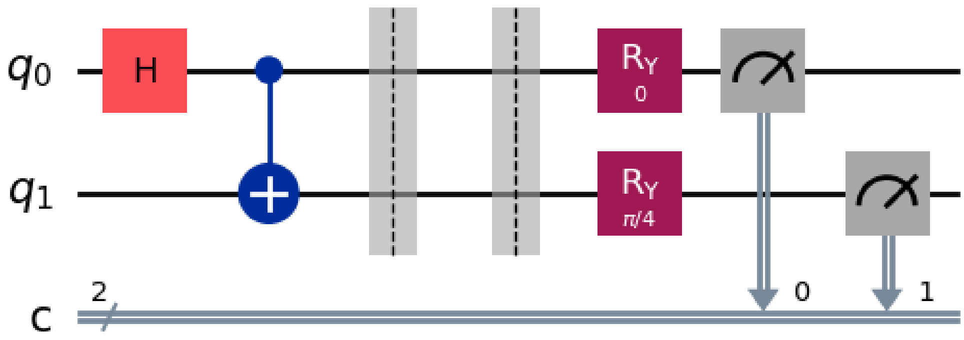
Here are the four possible circuits, depending on which questions are asked.

[6]:

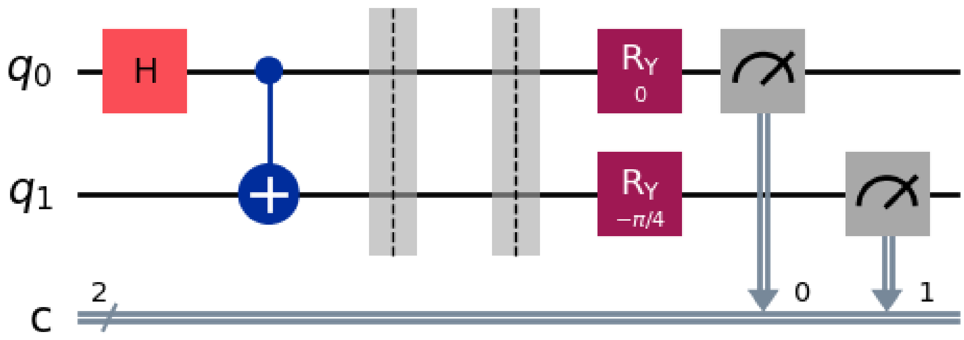
```
# Draw the four possible circuits
# Replace the ?

print("(x,y) = (0,0)")
display(chsh_circuit(0, 0).draw('mpl'))

print("(x,y) = (0,1)")
display(chsh_circuit(0, 1).draw('mpl'))

print("(x,y) = (1,0)")
display(chsh_circuit(1, 0).draw('mpl'))
```

```
print("(x,y) = (1,1)")
display(chsh_circuit(1, 1).draw('mpl'))
```

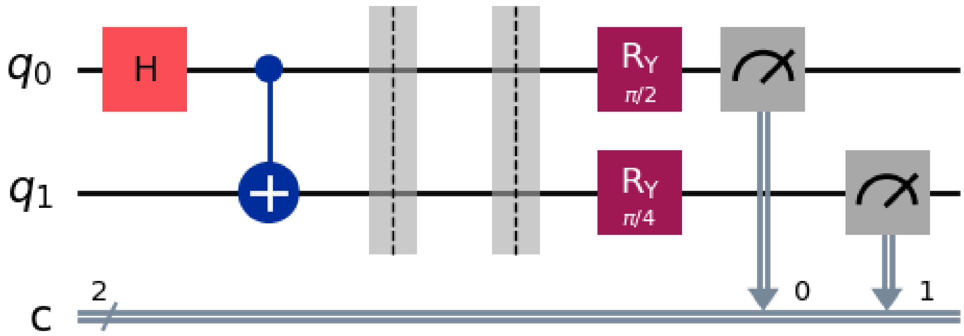
```
(x,y) = (0,0)
```



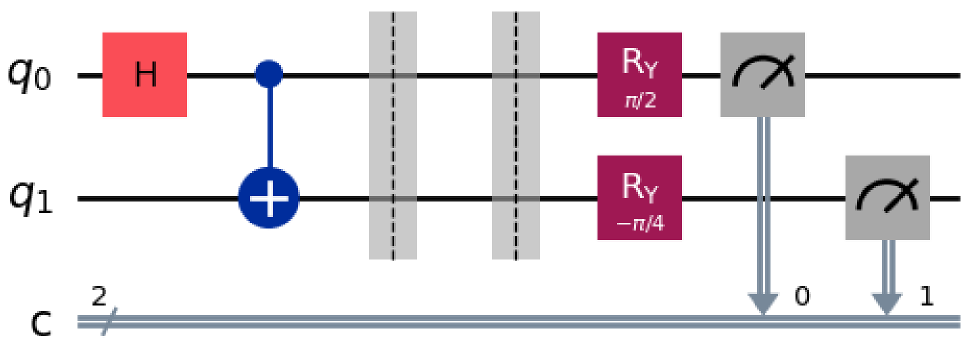

```
(x,y) = (0,1)
```




```
(x,y) = (1,0)
```




```
(x,y) = (1,1)
```




Now we'll create a job using the Aer simulator that runs the circuit a single time for a given input pair (x,y).

[7]:
```
sampler = Sampler()

def quantum_strategy(x, y):
    """Carry out the best strategy for the CHSH game.
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        (int, int): Alice and Bob's answer bits (respectively)
    """
    # Replace ?
```

```
    # `shots=1` runs the circuit once
    result = sampler.run(chsh_circuit(x, y), shots=1).result()
    statistics = result.quasi_dists[0].binary_probabilities()
    bits = list(statistics.keys())[0]
    a, b = bits[0], bits[1]
    return a, b
```

Finally, we'll play the game 1,000 times and compute the fraction of them that the strategy wins.

[8]:
```
# Replace ?
NUM_GAMES = 1000
TOTAL_SCORE = 0

for _ in range(NUM_GAMES):
    TOTAL_SCORE += chsh_game(quantum_strategy)

print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)
```

```
Fraction of games won: 0.856
```

We can also define a classical strategy and see how well it works. Feel free to change the code to try out different strategies!

[9]:
```
def classical_strategy(x, y):
    """An optimal classical strategy for the CHSH game
    Args:
        x (int): Alice's bit (must be 0 or 1)
        y (int): Bob's bit (must be 0 or 1)
    Returns:
        (int, int): Alice and Bob's answer bits (respectively)
    """
    # Alice's answer
    if x == 0:
        a = 0
    elif x == 1:
        a = 1

    # Bob's answer
    if y == 0:
        b = 1
    elif y == 1:
        b = 0

    return a, b
```

Again let's play the game 1,000 times to see how well it works.

[11]:
```python
# Replace ?
NUM_GAMES = 1000
TOTAL_SCORE = 0

for _ in range(NUM_GAMES):
    TOTAL_SCORE += chsh_game(classical_strategy)

print("Fraction of games won:", TOTAL_SCORE / NUM_GAMES)
```

```
Fraction of games won: 0.749
```

Although there's randomness involved, the statistics are very unlikely to deviate too much after 1,000 runs. The quantum strategy wins about 85% of the time while a classical strategy can't win more than about 75% of the time.

# 3 End of Notebook