Computer modeling of physical phenomena

Lab XII: Quantum Prisoner's Dilemma

Qiskit tutorial

Qiskit

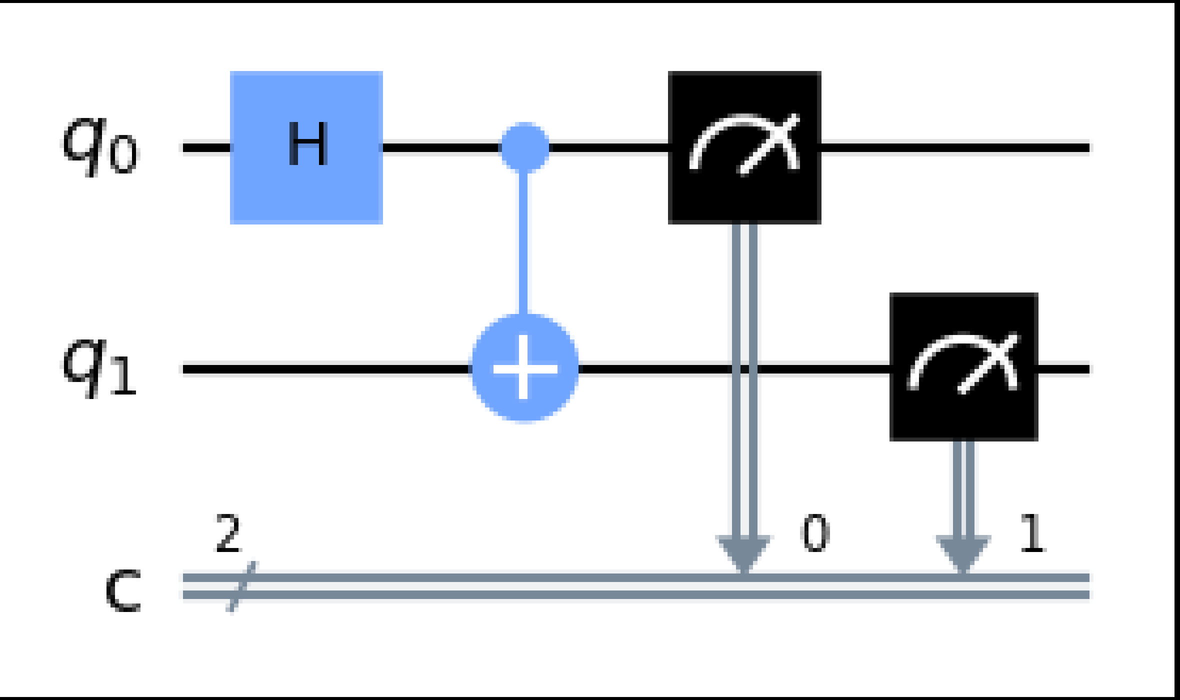
Elements for building a quantum future

Qiskit circuit

input

single/multi qubit operations

measurement



Qubit

```
from qiskit import QuantumRegister, ClassicalRegister
```

quantum bits number

```
q = QuantumRegister(2, 'qubit’)
```

bit name

```
c = ClassicalRegister(2, 'bit’)
```

classical bits number

$$\Psi = |00\rangle$$

qubits are defaultly initialized in state $|0\rangle$

Circuit

```python
from qiskit import QuantumCircuit
```

registers used in circuit

```python
circuit = QuantumCircuit(q, c)
```

single-qubit gate

target qubits

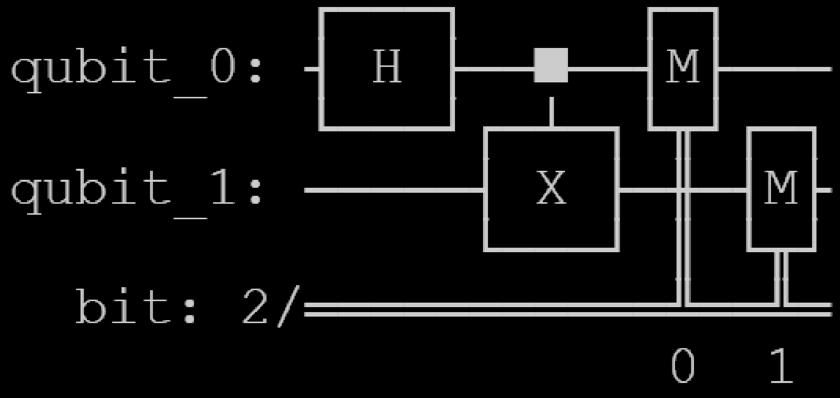
```python
circuit.h(q[0])
circuit.cnot(q[0], q[1])
```

multi-qubit gate

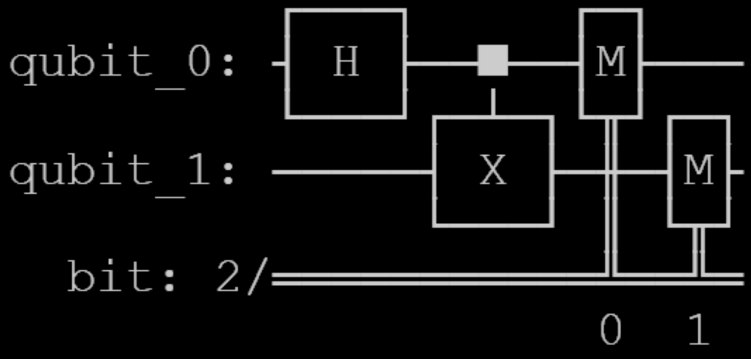
collapse qubits into classical bits

```python
circuit.measure(q, c)
circuit.draw()
```

visualize the circuit



Simulator




```
from qiskit import Aer, execute
```

type of simulator

```
backend = Aer.get_backend('statevector_simulator')
job = execute(circuit, backend)
res = job.result()
```

run simulation and get results

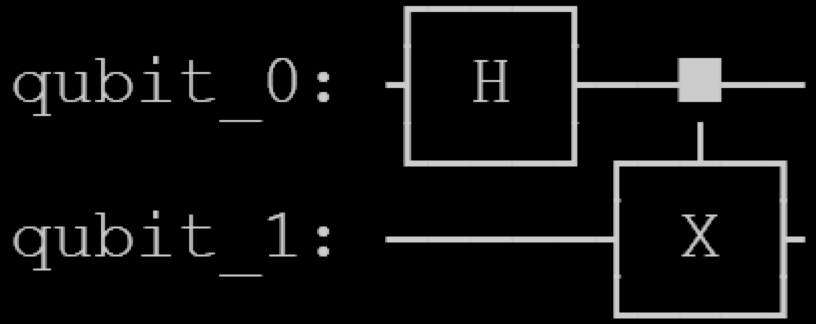
```
psi = res.get_statevector()
```

$\Psi = ?$

wavefunction amplitudes in format

$[\,|00\rangle, |10\rangle, |01\rangle, |11\rangle\,]$

Simulator (2)




```
circuit = QuantumCircuit(q)

circuit.h(q[0])
circuit.cnot(q[0], q[1])
#circuit.measure(q, c)
```

no measurement this time

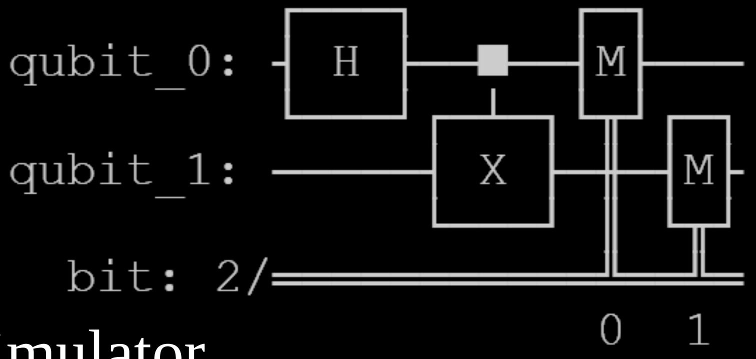
```
backend = Aer.get_backend('statevector_simulator')
job = execute(circuit, backend)
res = job.result()
psi = res.get_statevector()
```

$$\Psi = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

```
Statevector([0.70710678+0.j, 0.
+0.j, 0. +0.j, 0.70710678+0.j],
dims=(2, 2))
```

wavefunction amplitudes in format
$[\,|00\rangle, |10\rangle, |01\rangle, |11\rangle\,]$

Simulator (3)



To make proper measurements, we need to use another simulator.

```
backend = Aer.get_backend('qasm_simulator')
job = execute(circuit, backend, shots = 1000)
res = job.result()


prob = res.get_counts(circuit)
```

```
{'00': 527, '11': 497}
```

counts in format

$\{\texttt{'00'}: \alpha, \texttt{'01'}: \beta, \texttt{'10'}: \gamma, \texttt{'11'}: \delta\}$

$$\Psi = \alpha|00\rangle + \beta|10\rangle + \gamma|01\rangle + \delta|11\rangle)$$

qubits in dict are reversed

Task 1: Perfect Coin 0.5p

”Bonnie and Clyde have recently gotten into an argument about the philosophy of picking the correct side of a coin flip. Bonnie was raised by the moto ”Tails Never Fails”, while Clyde was taught ”Tails Always Fails”. Clyde suggests that they solve their disagreement with a series of coin flips, but Bonnie doesn’t trust any coin that Clyde owns, and vice versa for Clyde. Thus, they agreed to use a qubit as their coin.”

Use Qiskit and force Quantum Mechanics to once and for all decide, which ”Tails Philosophy” is real…

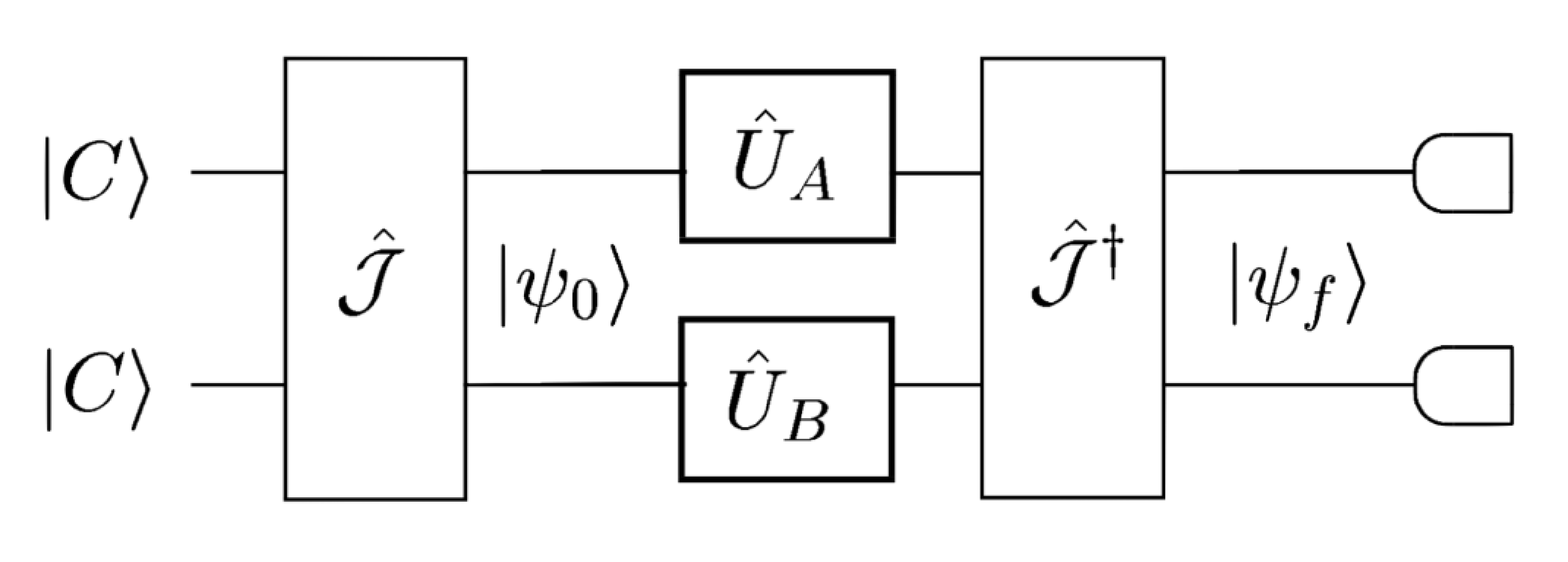
…at least in terms of local realism.

Design the circuit, run a simple simulation with n = 100 measurements and print the counts.

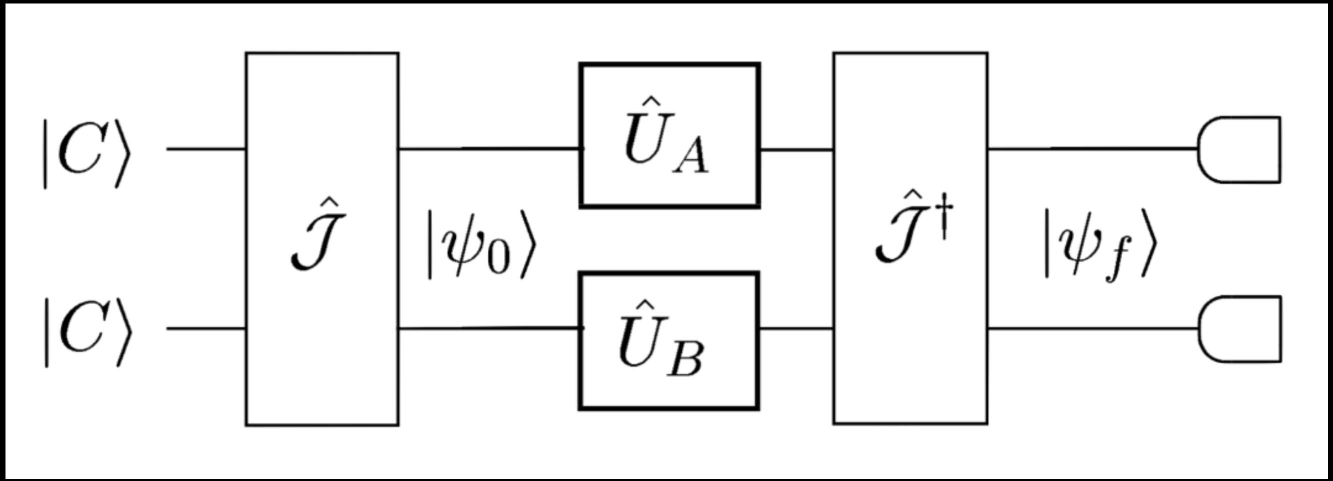
It’s as easy as you think.

Task 2: QPD 0.5p

Build QPD circuit based on the scheme below. Compare how the Nash equilibrium strategies work for different values of entanglement.



QPD operators



basis vectors

$$|C\rangle = \begin{pmatrix}1\\0\end{pmatrix}$$

$$|D\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

preparation step (generating γ-dependent entanglement)

$$\hat{J} = \exp(-i\,\gamma\,\hat{D}\otimes\hat{D}\,/\,2)$$

unitary operations

$$\hat{D} = \begin{pmatrix}0 & 1\\-1 & 0\end{pmatrix}$$

$$\hat{Q} = \begin{pmatrix}i & 0\\0 & -i\end{pmatrix}$$

Payoff

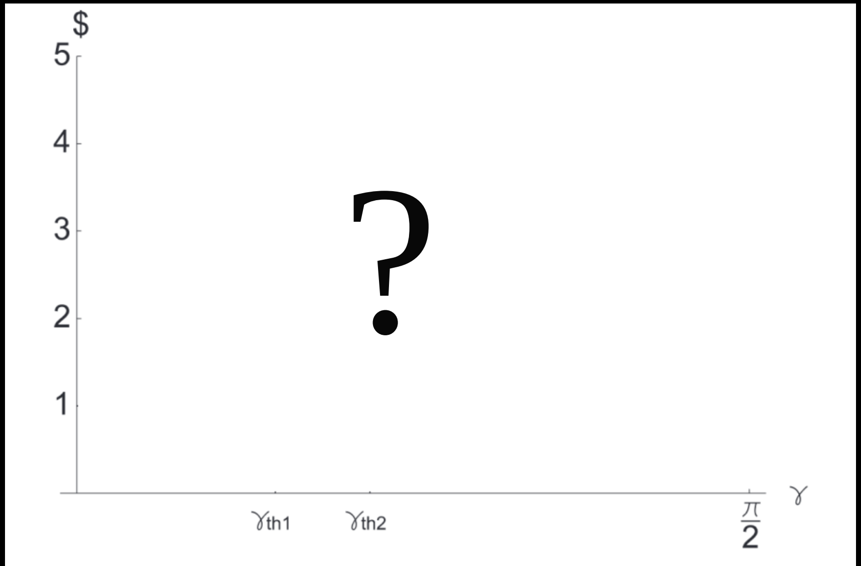
Player 1 \ Player 2	cooperate	defect
cooperate	(3, 3) r, r reward	(0, 5) s, t sucker's payoff, temptation
defect	(5, 0) t, s temptation, sucker's payoff	(1, 1) p, p punishment

probability of
a given outcome

$$\$_1 = r\, P_{CC} + t\, P_{DC} + s\, P_{CD} + p\, P_{DD}$$

Task 2: Details

Plot the payoff of one of the players for a range of entanglement parameters $\gamma \in [0, \pi/2]$ (at least 100 samples) and a set of strategies: $\hat{D} \times \hat{D}$, $\hat{D} \times \hat{Q}$, $\hat{Q} \times \hat{D}$, $\hat{Q} \times \hat{Q}$. You should find some interesting features of the plot for $\gamma_{th1} = \arcsin\left(\sqrt{1/5}\right)$ and $\gamma_{th2} = \arcsin\left(\sqrt{2/5}\right)$.



Task 2: Tips

You can build the entanglement matrix $\hat{J}$ in two ways: represent it as a series of standard single and multi-qubit operations (and then just add them in a correct order to the circuit) or build your own custom gate. Here are the tips for the second method.

```
from qiskit.extensions import UnitaryGate
J  =  ...  #  you  may  find  scipy.linalg.expm
           # useful, as well as np.kron
gate = UnitaryGate(J)
circuit.append(gate, q)
```

Extra task 0.2p

Run the QPD scheme for $\hat{Q} \times \hat{Q}$ strategy and few values of γ in a real quantum circuit. You can connect your Python code to it using a validation token or just build it manually at IBM website:

https://quantum-computing.ibm.com/composer/files/new

In both cases, you need to represent all the QPD quantum gates in terms of the available IBM operators, which may require a bit of calculation. Compare the IBM count rates with the ones produced by our Python simulator.