Quantized Games - Using Quantum Computing to influence the outcome of classical non-local games in fair or unfair way

November 3, 2025

] Sayan Samanta

1 Introduction

The concept of Quantized Games emerged when classical game theory intersected with Quantum Computing and Quantum Mechanics. In 1999, David A. Mayer was the first person to bring up the concept of Quantized games using PQ Penny Flip game. Since then, many classical games have been "quantized," meaning they incorporate either complete or partial quantum strategies to influence the outcomes of non-local games. These quantized games vary in complexity, leveraging the power of quantum computing. Quantum game theory takes advantage of fundamental quantum concepts such as superposition, entanglement. It also uses different quantum gates to form the quantum strategy which determines the outcome of the game. In this work, we will explore both the PQ Penny Flip game and a quantized card game to demonstrate how quantum strategies can provide advantages for specific players.

2 Area of Application

Quantum strategy in classical games finds the best applications in the games when they are zero sum and non-local and the games are in Nash equilibrium.

Zero-Sum Game: A zero-sum game is where a player can only gain by making the other player lose. This means that if player A and B is playing the game, then when A wins B can loses and vice-a-versa. They both can not gain together in a single move.

Non-local Game: In non-local games two cooperating players (Alice and Bob) receive questions from a referee and respond with answers based on incomplete information. The referee randomly selects questions according to a known distribution. Based on Alice and Bob's answers, the referee decides whether they win or lose.

Nash Equilibrium: Nash Equilibrium is a strategic profile where neither player can benefit by unilaterally deviating from their chosen strategies. The players of a game is considered to be in Nash Equilibrium if they have some predefined set of actions to be performed that is agreed upon. Only one player can not gain or win by deviating from those actions if the other player does not deviate from that.

The PQ Penny Flip game and Quantized Card game are in the same class.

3 PQ Penny Flip Game

3.1 Classical Game

In the classical version of PQ Penny Flip game, we have four steps. The game has two players P and Q and a referee. Both the players has only two sets of moves: flip(F) or no-flip(NF); and can make only one move at a time and the choice is random.

1. The referee keeps a penny heads-up in a black box so that none of the players can see the state of the penny until referee reveals it.
2. Q makes the first move with anyone of its move: F or N.
3. P makes the next move out of F and N.
4. Q makes the last move.

The referee now reveals the state of the penny. If the penny is heads-up, Q wins with a score of 1 and if the penny is heads-down, P wins with a score of 1. The game can be expressed in the following truth table:

Move-Q	Move-P	Move-Q	P	Q
N	N	N	0	1
N	N	F	1	0
N	F	N	1	0
N	F	F	0	1
F	N	N	0	1
F	N	F	1	0
F	F	N	1	0
F	F	F	0	1

Thus at the end of the game both the players are tied with a score of 4 and that means the game is extremely fair with a winning probability of

$$p = 0.5$$

The truth table is true as both the players can make any one of the defined moves with equal probability and neither player can win over the other if it is played for a longer time. The Qiskit implementation of this is shown next where 0 represents a P win and 1 represents a Q win and 0 is considered heads-up and 1 is considered heads-down.

Qiskit Implementation:

[5]:
```
from numpy.random import randint
def pq_classical_game(strategy):
    """Plays the PQ Penny Flip game
    Args:
    strategy (callable): A function that takes three bits (as `int`s) and
    returns one bit (also as `int`s). The strategy must follow the
    rules of the PQ Penny Flip game.
    Initial state of coin: 0 (Heads - up)
    Q chooses to flip or no-flip.
```

```
    P chooses to flip or no-flip.
    Q chooses to flip or no-flip.
    Returns:
    int: 1 for a Q win, 0 for a P win
    """

    # P and Q chooses to flip = 1 or no-flip = 0 randomly
    q1, p, q2 = randint(0, 2), randint(0, 2), randint(0, 2)
    # Use strategy to choose a
    a = strategy(q1,p,q2)
    # Referee opens the box, if coin heads up the Q wins, else P wins
    if a==0:
        return 1 # Q wins
    return 0 # P wins
def classical_strategy(q1, p, q2):
 """An optimal classical strategy for the PQ Penny Flip game
 Args:
 x (int): P's choice (must be 0 or 1)
 Returns:
 (int): 1 for a Tail, 0 for a Head (result of Q's drawing)
(respectively)
Initial State = 0 (Heads)
 """
 # Result after Q's decision
 if q1 == 0:
     a = 0
 elif q1 == 1:
     a = 1
 # Result after P's decision
 if p == 0:
     a = a
 elif p == 1:
     a = 1 - a
 # Result after Q's decision
 if q2 == 0:
     a = a
 elif q2 == 1:
     a = 1 - a
 return a
```

[9]:
```
NUM_GAMES = 1000
TOTAL_SCORE = 0
for _ in range(NUM_GAMES):
    TOTAL_SCORE += pq_classical_game(classical_strategy)
print("Fraction of games won by Q:", TOTAL_SCORE / NUM_GAMES)
print("Fraction of games won by P:", 1 - (TOTAL_SCORE / NUM_GAMES))
print("The game is fair to both players in classical version")
```

```
Fraction of games won by Q: 0.494
Fraction of games won by P: 0.506
The game is fair to both players in classical version
```

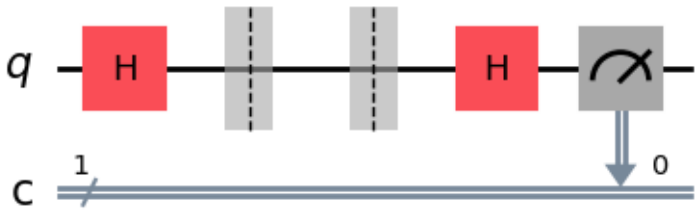
3.2 Quantized Game

In the Quantized version of the game, we make the game unfair to P by making Q win always regardless of the move P makes. Here, a heads-up is represented by $|0\rangle$ and a heads-down is represented by $|1\rangle$.

1. Q replaces its 1st move with a Hadamard(H) Gate.
2. P replaces its moves: flip is now replaced with X gate and no-flip is replaced with Identity.
3. Q replaces its 2nd move again with a H gate.

As the default state of q in Qiskit is $|0\rangle$, we can assume that the coin is in heads-up state. The circuit of this scheme can be shown with the following diagrams:

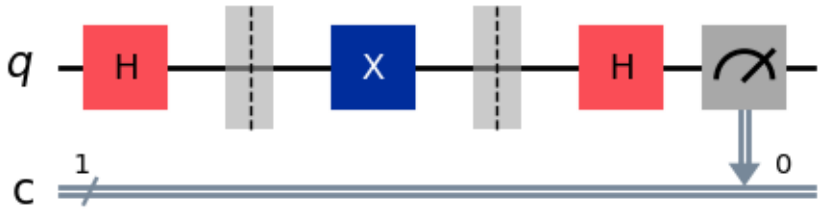
`(p = no-flip)`



Mathematical motivation behind the above scheme:

$$c = HIH|0\rangle$$
$$c = HI|+\rangle$$
$$c = H|+\rangle$$
$$c = |0\rangle$$

`(p = flip)`



Mathematical motivation behind the above scheme:

$$c = HXH|0\rangle$$
$$c = HX|+\rangle$$
$$c = H|+\rangle$$
$$c = |0\rangle$$

Thus, we can see that we always measure 0 regardless of the move P makes and we represent heads-up with 0 and heads-down with 1. So, Q always wins here (Our implementation). If consideration is altered, then P always wins in that case.

Qiskit Implementation:

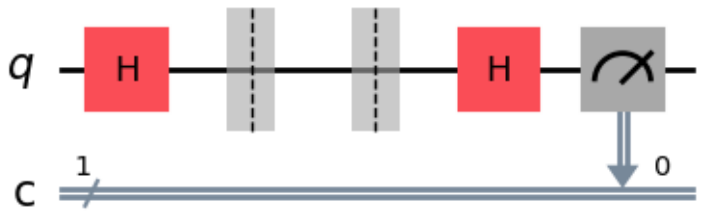
[10]:
```
#Quantized PQ game
from qiskit import QuantumCircuit
from qiskit_aer.primitives import Sampler
from numpy import pi
from numpy.random import randint
from numpy.random import randint
def pq_quantized_game(strategy):
    """Plays the PQ Penny Flip game
    Args:
    strategy (callable): A function that takes one bit (as `int`s) and
    returns one bit (also as `int`s). The strategy must follow the
    rules of the PQ Penny Flip game.
    Initial state of coin: 0 (Heads - up)
    Q chooses to flip or no-flip. (Always : H gate)
    P chooses to flip or no-flip. (flip = X gate and no-flip = I)
    Q chooses to flip or no-flip. (Always : H gate)
    Returns:
    int: 1 for a Q win, 0 for a P win
    """

    # P and Q chooses to flip = 1 or no-flip = 0 randomly
    p = randint(0, 2)
```

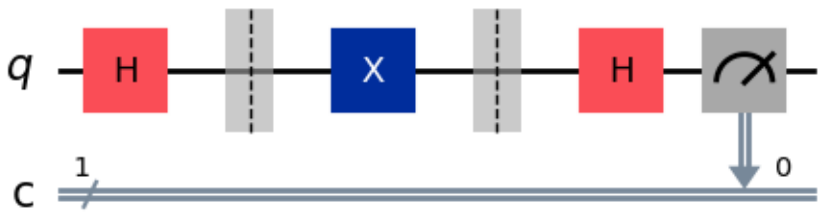
```
    # Use strategy to choose a
    a = strategy(p)
    # Referee opens the box, if coin heads up the Q wins, else P wins
    if a==0:
        return 1 # Q wins
    return 0 # P wins
```

[11]:
```
def pq_circuit(p):
    """Creates a `QuantumCircuit` that implements the best PQ Penny Flip␣
 ↪strategy.
    Args:
    p (int): P's decision (must be 0 or 1)
    Returns:
    QuantumCircuit: Circuit that, when run, returns Alice and Bob's
    answer bits.
    """
    qc = QuantumCircuit(1, 1)
    qc.h(0) #Q always apply H gate
    #qc.cx(0, 1)
    qc.barrier()
    # P's decision to flip or no-flip
    # Flip = X gate and no-flip= I
    if p == 1:
        qc.x(0)
    qc.barrier()
    qc.h(0) #Q always apply H gate
    qc.measure(0, 0)
    return qc
# Draw the two possible circuits
print("(p = 0)")
display(pq_circuit(0).draw('mpl'))
print("(p = 1)")
display(pq_circuit(1).draw('mpl'))
```

```
(p = 0)
```



(p = 1)



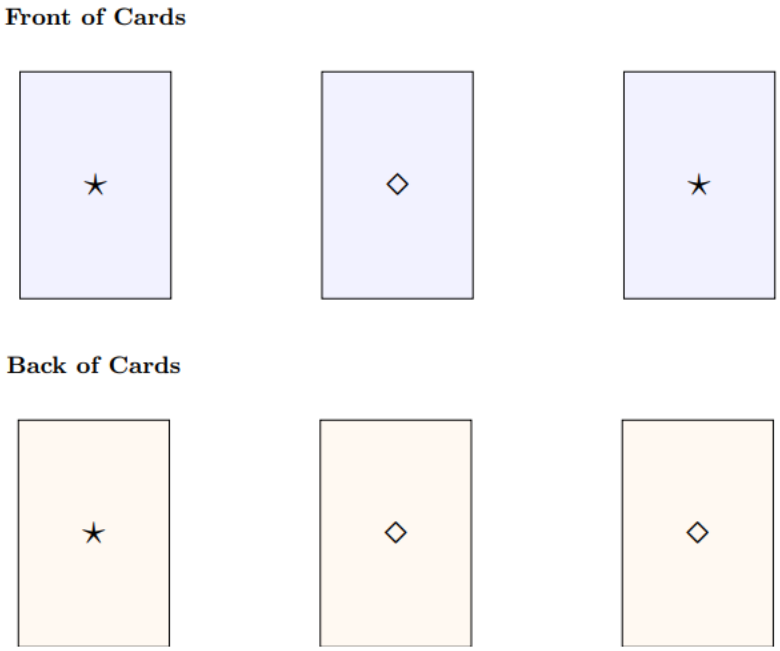
[12]:
```
sampler = Sampler()
def quantum_strategy(p):
    """Carry out the best strategy for the PQ Penny Flip game.
    Args:
    p (int): P's decision bit (must be 0 or 1)
    Returns:
    (int): status bits (0 for heads, 1 for tail)
    """
    # `shots=1` runs the circuit once
    result = sampler.run(pq_circuit(p), shots=1).result()
    statistics = result.quasi_dists[0].binary_probabilities()
    bits = list(statistics.keys())[0]
    a = int(bits[0])
    return a
NUM_GAMES = 1000
TOTAL_SCORE = 0
for _ in range(NUM_GAMES):
    TOTAL_SCORE += pq_quantized_game(quantum_strategy)
print("Fraction of games won by Q:", TOTAL_SCORE / NUM_GAMES)
print("Fraction of games won by P:", 1 - (TOTAL_SCORE / NUM_GAMES))
print("Thus the game has been made unfair to P as Q always wins with Quantized␣
 ↪Strategy")
```

```
Fraction of games won by Q: 1.0
Fraction of games won by P: 0.0
Thus the game has been made unfair to P as Q always wins with Quantized Strategy
```

4 Bob-Alice Card Game

4.1 Classical Game

In the classical version of the game, Alice and Bob has three cards with the following markings: one card with a star on both sides, another card with a diamond on both sides and the last card has star on one side and diamond on the other side.



The game has the following steps:

1. A referee puts the cards in a black box and shakes it up to randomize their position and top side.
2. Bob now draws a card from the black box without flipping such that both the players can see the only top side of the card.
3. Now referee checks both sides of the card and if the card has identical markings on both side, Alice wins. If the card has different markings on both side Bob wins.
4. Bob also has an option to quit after drawing the card but that does not really help him.

It is very evident that Alice has a winning probability of $P(A) = \frac{2}{3}$ and Bob has a winning probability of $P(B) = \frac{1}{3}$. Thus the game clearly unfair to Bob.

Qiskit Implementation:

[13]:
```
#Create a function to swap elements in tuple
#May be used in later stages
def swap_elements(tup, idx1, idx2):
    lst = list(tup)
    lst[idx1], lst[idx2] = lst[idx2], lst[idx1]
```

```
    return tuple(lst)

# Example usage:
my_tuple = (1, 2, 3)
new_tuple = swap_elements(my_tuple, 1, 2)
print(new_tuple)
```

```
(1, 3, 2)
```

[14]:
```
from numpy.random import randint
def card_classical_game(strategy):
    """Alice and Bob Plays the Classical Card game
    Args:
    strategy (callable): A function that takes three bits (as `int`s) and
    returns one bit (also as `int`s). The strategy must follow the
    rules of the Classical Card Game.
    Three cards:
    S has star on both side. (0)
    D has diamond on both side. (1)
    M has star on one side and diamond on other side. (2)
    Alice puts it in a black box and shakes it.
    Bob randomly draws a card.
    If that card has same marking on both side Alice wins, otherwise Bob.
    Returns:
    int: 1 for a Bob win, 0 for a Alice win
    """
    #Box with three cards
    s, d, m = 0, 1, 2
    # Use strategy to choose a
    a, b, c = strategy(s,d,m)
    # Bob randomly picks a card after Alice shakes it
    lst = (a, b, c)
    d = randint(0, 3)
    tpl = list(lst)
    if tpl[d]==2: #If different on both sides
        return 1 # Bob wins
    return 0 # Alice wins
```

[15]:
```
def classical_strategy(s, d, m):
 """An optimal classical strategy for the Alice and Bob Classical card game
 Args:
 s (int): denoted by 0
 d (int): denoted by 1
 m (int): denoted by 2
 Returns:
 (int, int, int): Randomly swaps the elements resembles shaking of the box
 """
 id1 = randint(0, 3)
```

```
    id2 = randint(0, 3)
    a, b, c = swap_elements((s, d, m), id1, id2) #Shakes the black box
    return a, b, c
```

[16]:
```
NUM_GAMES = 1000
TOTAL_SCORE = 0
for _ in range(NUM_GAMES):
    TOTAL_SCORE += card_classical_game(classical_strategy)
print("Fraction of games won by Bob:", TOTAL_SCORE / NUM_GAMES)
print("Fraction of games won by Alice:", 1 - (TOTAL_SCORE / NUM_GAMES))
print("The Game is biased Bob with winning probability of 1/3")
```

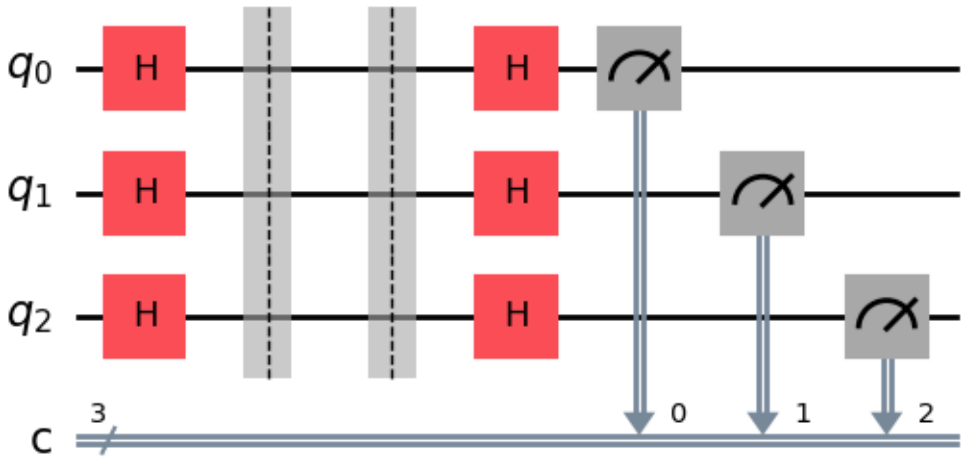
```
Fraction of games won by Bob: 0.321
Fraction of games won by Alice: 0.679
The Game is biased Bob with winning probability of 1/3
```

4.2 Quantized Game

In Quantized version of this game, the star marking is represented by $|0\rangle$ and diamond marking is represented by $|1\rangle$. Once the referee shakes up the black box to randomize the cards, the state of the top side of the cards inside the box can be represented $|r\rangle$ as set of three qubits $|r\rangle = |r_0, r_1, r_2\rangle$ where $r_k \in (0, 1)$ and $k \in (0, 1, 2)$.

Now, we allow Bob to make a quantum query machine that depends on state $|r_k\rangle$ in the box. The query machine is constructed such that if $|r_k\rangle = |0\rangle$ the final Unitary Matrix is $U = HIH$ and if $|r_k\rangle = |1\rangle$ the final Unitary Matrix is $U = HZH$ for each of the qubit and the inputs to the query machine are always $|0\rangle$. This can be depicted by the following circuits:

$$|r\rangle = |0\rangle, |0\rangle, |0\rangle$$

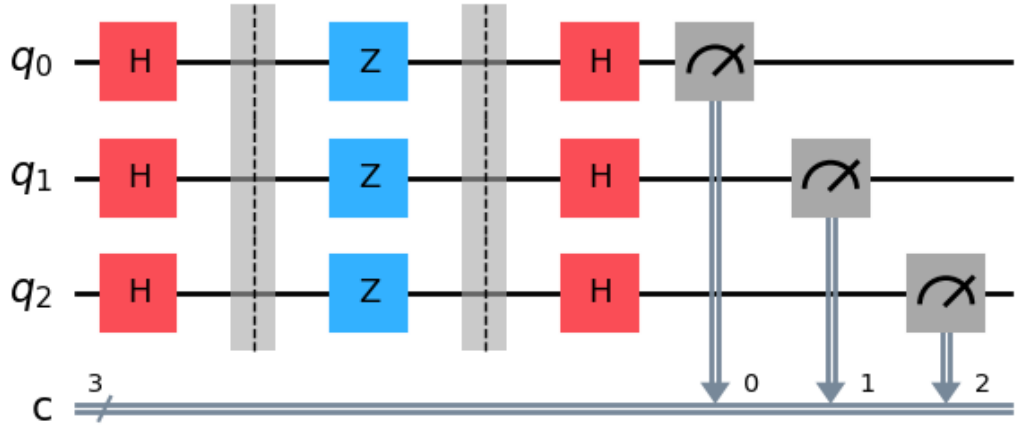


The mathematical motivation behind the above scheme:

$$|r_k\rangle = HIH|0\rangle$$
$$|r_k\rangle = HI|+\rangle$$
$$|r_k\rangle = H|+\rangle$$
$$|r_k\rangle = |0\rangle$$

We will get 0 when measured which means that the state of the top of that card is star marked.

$$|r\rangle = |1\rangle, |1\rangle, |1\rangle$$



The mathematical motivation behind the above scheme:

$$|r_k\rangle = HZH|0\rangle$$
$$|r_k\rangle = HZ|+\rangle$$
$$|r_k\rangle = H|-\rangle$$
$$|r_k\rangle = |1\rangle$$

We will get 1 when measured which means that the state of the top of that card is diamond marked. So at any point of time the query can result two different sets which represents only the state of top side of the cards:

$$S_0 = \{|0\rangle, |0\rangle, |1\rangle\}$$
$$S_1 = \{|0\rangle, |1\rangle, |1\rangle\}$$

This is because star marked card has $|0\rangle$ on both sides, diamond marked card has $|1\rangle$ on both sides and the other card has marked card has $|0\rangle$ on one side and $|1\rangle$ on the other. Now, Bob draws a card randomly from the black box and the result of the game still governs by the same set of rules i.e. if the card has identical markings on both side, Alice wins and if the card has different markings

on both side Bob wins. But he has the information of the state of top side of the cards in the black box. If the drawn card has star marking (state-0) on top of the card and the query machine results in S_0 then that Bob has a 50% chance of winning and if the drawn card has diamond marking (state-1) on top of the card and the query machine results in S_1 then that Bob has a 50% chance of winning. Bob quits the game in any other cases.

Thus using this quantum query machine, the result of the game is made fair from the classical unfair game. Now Alice has a winning probability of $P(A) = \frac{1}{2}$ and Bob has a winning probability of $P(B) = \frac{1}{2}$.

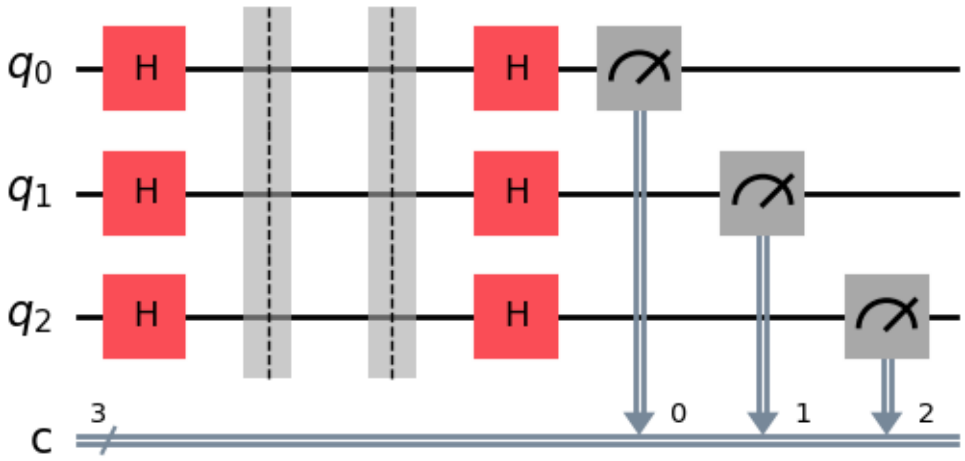
Qiskit Implementation:

[17]:
```
#Quantized Alice and Bob Card game
from numpy.random import randint
def card_quantized_game(strategy):
    """Alice and Bob Plays the Classical Card game
    Args:
    strategy (callable): A function that takes three bits (as `int`s) and
    returns one bit (also as `int`s). The strategy must follow the
    rules of the Classical Card Game.
    Three cards:
    S has star on both side. (0)
    D has diamond on both side. (1)
    M has star on one side and diamond on other side. (0 or 1)
    Alice puts it in a black box and shakes it.
    Bob randomly draws a card.
    If that card has same marking on both side Alice wins, otherwise Bob.
    Returns:
    int: 1 for a Alice and Bob wins 50%, 0 for Bob quit
    """
    #Box with three cards
    s, d, m = 0, 1, randint(0, 2)
    # Use strategy to choose a
    a, b, c = strategy(s,d,m)
    # Bob randomly picks a card after Alice shakes it
    lst = (a, b, c)
    d = randint(0, 3)
    tpl = list(lst)
    if (m==0) & (tpl[d]==0): #Randomly picks the card
        return 1 # Bob has 50% chance of winning
    elif (m==1) & (tpl[d]==1): #Randomly picks the card
        return 1 # Bob has 50% chance of winning
    return 0; #Bob quits
```

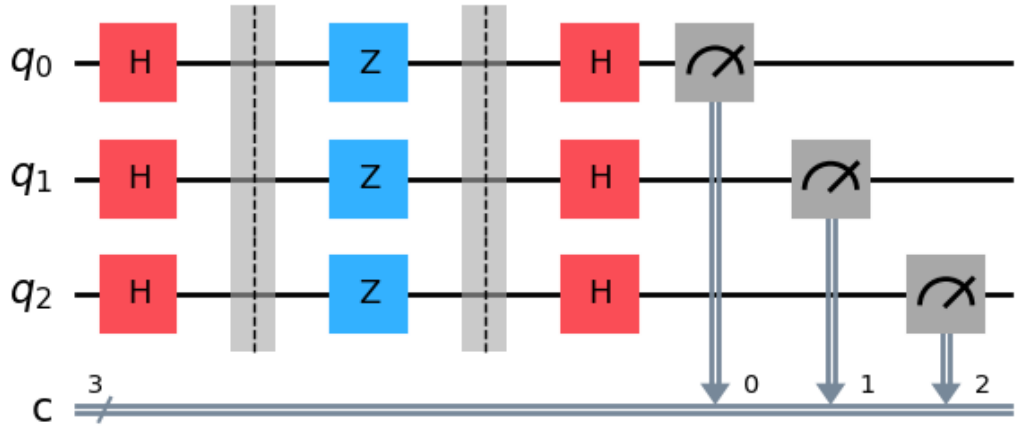
[18]:
```
from qiskit import QuantumCircuit
from qiskit_aer.primitives import Sampler
from numpy import pi
```

```
from numpy.random import randint
def card_circuit(s, d, m):
    """Creates a `QuantumCircuit` that implements the best CHSH strategy.
    Args:
    s, d, m (int, int, int): Three Cards
    Returns:
    QuantumCircuit: Circuit that, when run, returns Alice and Bob's
    answer bits.
    """
    qc = QuantumCircuit(3, 3)
    qc.h(0)
    qc.h(1)
    qc.h(2)
    qc.barrier()
    if s == 1:
        qc.z(0)
    if d == 1:
        qc.z(1)
    if m == 1:
        qc.z(2)
    qc.barrier()
    qc.h(0)
    qc.h(1)
    qc.h(2)
    qc.measure(0, 0) #query for 1st card
    qc.measure(1, 1) #query for 2nd card
    qc.measure(2, 2) #query for 3rd card
    return qc
# Draw the few possible circuits
print("(s,d,m) = (0,0,0)")
display(card_circuit(0, 0, 0).draw('mpl'))
print("(s,d,m) = (1,1,1)")
display(card_circuit(1, 1, 1).draw('mpl'))
```

```
(s,d,m) = (0,0,0)
```




```
(s,d,m) = (1,1,1)
```



[19]:
```
sampler = Sampler()
def quantum_strategy(s, d, m):
    """An optimal classical strategy for the Alice and Bob Classical card game
    Args:
    s (int): denoted by 0
    d (int): denoted by 1
    m (int): denoted by 2
    Returns:
```

```
    (int, int, int): Randomly swaps the elements resembles shaking of the box
    """
    id1 = randint(0, 3)
    id2 = randint(0, 3)
    a1, b1, c1 = swap_elements((s, d, m), id1, id2) #Shakes the black box
    # `shots=1` runs the circuit once
    result = sampler.run(card_circuit(a1, b1, c1), shots=1).result()
    statistics = result.quasi_dists[0].binary_probabilities()
    bits = list(statistics.keys())[0]
    a, b, c = int(bits[0]), int(bits[1]), int(bits[2]) #quantum querry
    return a, b, c
```

[20]:
```
NUM_GAMES = 1000
TOTAL_SCORE = 0
GAMES_PLAYED = 0
for _ in range(NUM_GAMES):
    value=card_quantized_game(quantum_strategy)
    if value==1:
        GAMES_PLAYED += 1 #Games that Bob did not quit
        TOTAL_SCORE += value
print("Fraction of games won by Bob:", (TOTAL_SCORE / GAMES_PLAYED)/2) #Divide␣
 ↪by 2 for 50% chance
print("Fraction of games won by Alice:", 1 - (TOTAL_SCORE / GAMES_PLAYED)/2)
print("Thus the game has been made fair to both")
```

```
Fraction of games won by Bob: 0.5
Fraction of games won by Alice: 0.5
Thus the game has been made fair to both
```

5 Conclusion

The PQ Penny Flip was the first ever classical game that was quantized. There are many classical games which were quantized since then and they also vary in complexity and layers of Quantum logic. The games discussed here shows how quantum strategy influence the results of the classical games in a desired way. The extremely fair Penny Flip game has been made unfair in the quantized version with Q always winning and the classical Bob-Alice card game which was unfair to Bob, has been made fair in the quantized version.

References

[1] H. Guo, J. Zhang, and G. J. Koehler, "A survey of quantum games," Decision Support Systems, vol. 46, no. 1, pp. 318–332, Dec. 2008, doi: 10.1016/j.dss.2008.07.001

[2] E. Price, "Quantum Games and Game Strategy". Available: https://homes.psd.uchicago.edu/ sethi/Teaching/P243-W2020/final-papers/price.pdf

[3] J. O. Grabbe, "An Introduction to Quantum Game Theory." arXiv, Jun. 27, 2005. Accessed: Jul. 11, 2024. [Online]. Available: http://arxiv.org/abs/quant-ph/0506219