Title : Coin Flipping instead of 3 puzzle problem

In [16]:
```
!pip install --upgrade qiskit
```

```
Requirement already satisfied: qiskit in /opt/conda/lib/python3.10/site-packa
ges (0.44.2)
Requirement already satisfied: qiskit-terra==0.25.2.1 in /opt/conda/lib/pytho
n3.10/site-packages (from qiskit) (0.25.2.1)
Requirement already satisfied: python-dateutil>=2.8.0 in /opt/conda/lib/pytho
n3.10/site-packages (from qiskit-terra==0.25.2.1->qiskit) (2.8.2)
Requirement already satisfied: scipy>=1.5 in /opt/conda/lib/python3.10/site-p
ackages (from qiskit-terra==0.25.2.1->qiskit) (1.9.3)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.10/site-
packages (from qiskit-terra==0.25.2.1->qiskit) (1.23.5)
Requirement already satisfied: typing-extensions in /opt/conda/lib/python3.1
0/site-packages (from qiskit-terra==0.25.2.1->qiskit) (4.5.0)
Requirement already satisfied: ply>=3.10 in /opt/conda/lib/python3.10/site-pa
ckages (from qiskit-terra==0.25.2.1->qiskit) (3.11)
Requirement already satisfied: psutil>=5 in /opt/conda/lib/python3.10/site-pa
ckages (from qiskit-terra==0.25.2.1->qiskit) (5.9.4)
Requirement already satisfied: dill>=0.3 in /opt/conda/lib/python3.10/site-pa
ckages (from qiskit-terra==0.25.2.1->qiskit) (0.3.7)
Requirement already satisfied: symengine<0.10,>=0.9 in /opt/conda/lib/python
3.10/site-packages (from qiskit-terra==0.25.2.1->qiskit) (0.9.2)
Requirement already satisfied: stevedore>=3.0.0 in /opt/conda/lib/python3.10/
site-packages (from qiskit-terra==0.25.2.1->qiskit) (4.1.1)
Requirement already satisfied: sympy>=1.3 in /opt/conda/lib/python3.10/site-p
ackages (from qiskit-terra==0.25.2.1->qiskit) (1.11.1)
Requirement already satisfied: rustworkx>=0.13.0 in /opt/conda/lib/python3.1
0/site-packages (from qiskit-terra==0.25.2.1->qiskit) (0.13.0)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-pac
kages (from python-dateutil>=2.8.0->qiskit-terra==0.25.2.1->qiskit) (1.16.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in /opt/conda/lib/python3.1
0/site-packages (from stevedore>=3.0.0->qiskit-terra==0.25.2.1->qiskit) (5.1
1.1)
Requirement already satisfied: mpmath>=0.19 in /opt/conda/lib/python3.10/site
-packages (from sympy>=1.3->qiskit-terra==0.25.2.1->qiskit) (1.3.0)

[notice] A new release of pip available: 23.1.1 -> 23.2.1
[notice] To update, run: pip install --upgrade pip
```

In [ ]:
```
from qiskit import Aer, transpile, assemble, QuantumCircuit
from qiskit.aqua import QuantumInstance
from qiskit.aqua.algorithms import QAOA
from qiskit.aqua.components.optimizers import COBYLA

# Define the objective function
def objective_function(x):
    cost = 0
    state = [[1, 2, 3], [4, 5, 6], [7, 8, None]]

    for i in range(3):
        for j in range(3):
            if state[i][j] is not None and state[i][j] != x[i][j]:
                cost += 1

    return cost

# Define the QAOA solver
def solve_3puzzle_qaoa():
    backend = Aer.get_backend('qasm_simulator')
    optimizer = COBYLA(maxiter=100)
    qaoa = QAOA(optimizer=optimizer, p=1, quantum_instance=QuantumInstance(back

    # Generate the initial state circuit
    initial_state = QuantumCircuit(9)
    for i in range(3):
        for j in range(3):
            initial_state.h(i * 3 + j)
    initial_state.barrier()

    # Generate the mixer circuit
    mixer = QuantumCircuit(9)
    mixer.cz(0, 1)
    mixer.cz(0, 3)
    mixer.cz(1, 2)
    mixer.cz(1, 4)
    mixer.cz(2, 5)
    mixer.cz(3, 4)
    mixer.cz(3, 6)
    mixer.cz(4, 5)
    mixer.cz(4, 7)
    mixer.cz(5, 8)
    mixer.cz(6, 7)
    mixer.cz(7, 8)

    # Solve the problem using QAOA
    qaoa.initial_state = initial_state
    qaoa.mixer = mixer
    qaoa.objective_function = objective_function

    result = qaoa.compute_minimum_eigenvalue()
    solution = result.x

    return solution

# Solve the 3-puzzle problem using QAOA
solution = solve_3puzzle_qaoa()
```

```
# Print the solution
print("Solution found!")
for i in range(0, 9, 3):
    print(solution[i:i + 3])
```

In [1]:

```
from qiskit import QuantumCircuit, Aer, execute
from qiskit.visualization import plot_histogram

# Define the initial state
initial_state = [2, 5, 3, 1, 8, 6, 4, 7, None]

# Define the goal state
goal_state = [1, 2, 3, 4, 5, 6, 7, 8, None]

# Define the quantum circuit
qc = QuantumCircuit(18, 18)

# Initialize the circuit with the initial state
for i, tile in enumerate(initial_state):
    if tile is not None:
        qc.x(i)

# Perform swaps to reach the goal state
for i, tile in enumerate(goal_state):
    if tile is not None:
        initial_index = initial_state.index(tile)
        target_index = goal_state.index(tile)
        qc.swap(i, initial_index + 9)
        qc.swap(i, target_index + 9)

# Measure the final state
qc.measure(range(9), range(9))

# Simulate the circuit
backend = Aer.get_backend('qasm_simulator')
job = execute(qc, backend, shots=1024)
result = job.result()
counts = result.get_counts(qc)

# Find the most probable state (solution)
max_count = max(counts.values())
solution = [state for state, count in counts.items() if count == max_count][0]

# Print the solution
print("Solution found!")
for i in range(0, 9, 3):
    print(solution[i:i + 3])
```

```
Solution found!
000
000
000
```

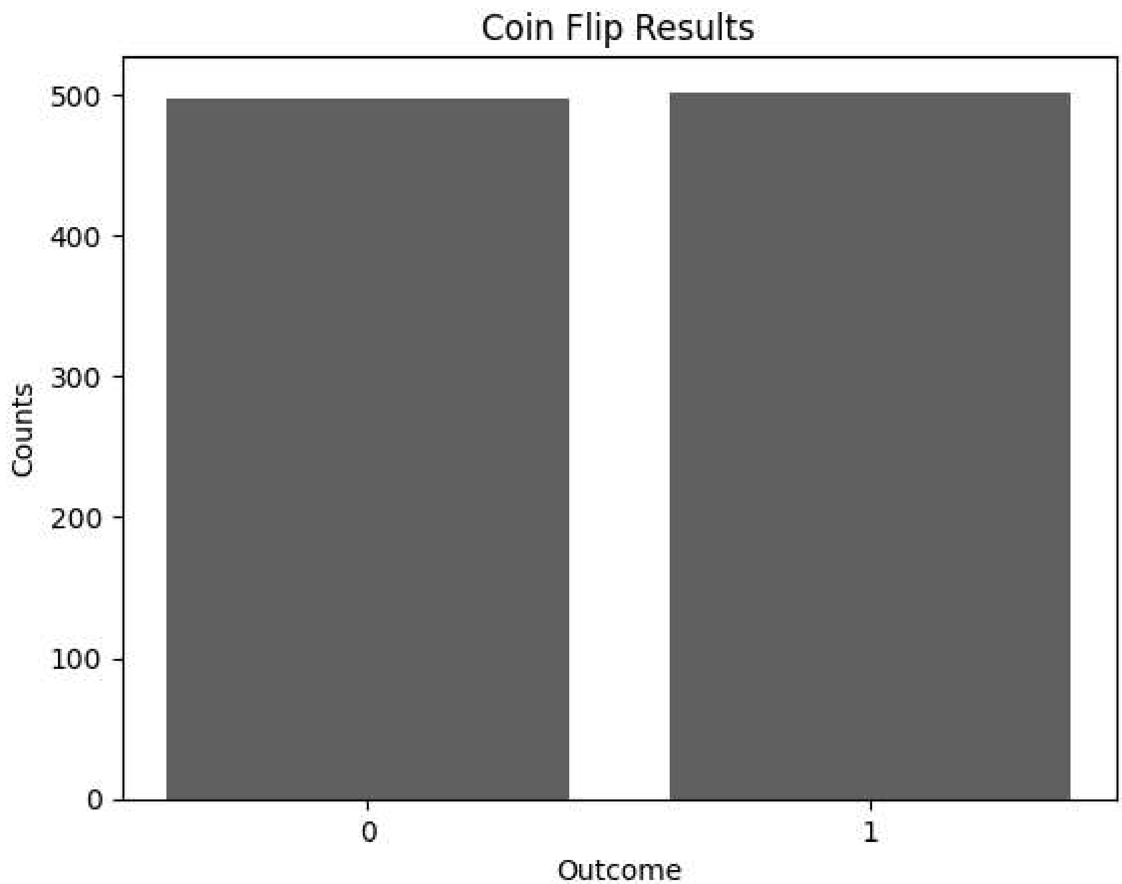
In [2]:
```python
from qiskit import Aer, execute, QuantumCircuit

# Define the quantum circuit
qc = QuantumCircuit(1, 1)
qc.h(0)  # Apply Hadamard gate for superposition
qc.measure(0, 0)  # Measure qubit and store result in classical bit

# Use the Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Set the number of shots (measurements)
num_shots = 1000

# Execute the quantum circuit on the simulator with multiple shots
job = execute(qc, simulator, shots=num_shots)

# Get the result of the measurements
result = job.result()
counts = result.get_counts()

# Print the coin flip results
print("Coin Flip Results:")
for outcome, count in counts.items():
    print(f"{outcome}: {count} ({count/num_shots*100:.2f}%)")
```

```
Coin Flip Results:
1: 540 (54.00%)
0: 460 (46.00%)
```

In [3]:
```python
from qiskit import Aer, execute, QuantumCircuit

# Define the quantum circuit
qc = QuantumCircuit(1, 1)
qc.h(0)  # Apply Hadamard gate for superposition
qc.measure(0, 0)  # Measure qubit and store result in classical bit

# Use the Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Set the number of shots (measurements)
num_shots = 1000

# Execute the quantum circuit on the simulator with multiple shots
job = execute(qc, simulator, shots=num_shots)

# Get the result of the measurements
result = job.result()
counts = result.get_counts()

# Print the coin flip results
print("Coin Flip Results:")
for outcome, count in counts.items():
    print(f"{outcome}: {count} ({count/num_shots*100:.2f}%)")
```

```
Coin Flip Results:
0: 488 (48.80%)
1: 512 (51.20%)
```

In [4]:
```
import matplotlib.pyplot as plt
from qiskit import Aer, execute, QuantumCircuit

# Define the quantum circuit
qc = QuantumCircuit(1, 1)
qc.h(0)  # Apply Hadamard gate for superposition
qc.measure(0, 0)  # Measure qubit and store result in classical bit

# Use the Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Set the number of shots (measurements)
num_shots = 1000

# Execute the quantum circuit on the simulator with multiple shots
job = execute(qc, simulator, shots=num_shots)

# Get the result of the measurements
result = job.result()
counts = result.get_counts()

# Plot the coin flip results
outcomes = list(counts.keys())
counts_list = list(counts.values())

plt.bar(outcomes, counts_list)
plt.xlabel('Outcome')
plt.ylabel('Counts')
plt.title('Coin Flip Results')
plt.show()
```




```
In [ ]:
```