Certainly, here is the complete markdown report with code sections and descriptions:

```
# Grover's Algorithm Report

## Part 1
### Introduction
Grover's algorithm is implemented in this section, a quantum algorithm designed to
search an unsorted database of items quadratically faster than classical
algorithms.

### Importing Libraries
The necessary libraries for implementing the algorithm are imported, including
numpy for numerical operations, Qiskit for quantum computing, and its relevant
components like QuantumCircuit, ClassicalRegister, QuantumRegister, and
visualization tools.

```python
import numpy as np
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, Aer,
execute, transpile, assemble
from qiskit.circuit.library import CU1Gate
from qiskit.visualization import *
```

`CCCZ` Function

The `CCCZ` function is defined to implement a Controlled-Controlled-Controlled-Z gate. It serves as a subroutine in the oracle and diffusion operators to perform controlled-Z operations with phase factors on the given quantum bits.

```
def CCCZ(circuit, quantum):
    cu1_positive = CU1Gate(theta= np.pi/4)
    cu1_negative = CU1Gate(theta=-np.pi/4)

    circuit.append(cu1_positive,[quantum[0], quantum[3]])
    circuit.cx(quantum[0], quantum[1])
    circuit.append(cu1_negative,[quantum[1], quantum[3]])
    circuit.cx(quantum[0], quantum[1])
    circuit.append(cu1_positive,[quantum[1], quantum[3]])
    circuit.cx(quantum[1], quantum[2])
    circuit.append(cu1_negative,[quantum[2], quantum[3]])
    circuit.cx(quantum[0], quantum[2])
    circuit.append(cu1_positive,[quantum[2], quantum[3]])
    circuit.cx(quantum[1], quantum[2])
    circuit.append(cu1_negative,[quantum[2], quantum[3]])
    circuit.cx(quantum[0], quantum[2])
    circuit.append(cu1_positive,[quantum[2], quantum[3]])
```

`oracle` Function
```

The `oracle` function is defined to create the oracle operator, which marks the target state in Grover's algorithm. It flips the phase of the target state by applying the `CCCZ` gate with suitable phase factors.

```
def oracle(circuit, quantum, target_state):
    for i, bit in enumerate(target_state):
        if bit == '0':
            circuit.x(quantum[i])

    CCCZ(circuit, quantum)

    for i, bit in enumerate(target_state):
        if bit == '0':
            circuit.x(quantum[i])

    circuit.barrier(quantum)
```

`diffusion` Function

This function is defined to construct the diffusion operator in Grover's algorithm. It inverts the amplitude of the state with the maximum probability, creating constructive interference to amplify the target state.

```
def diffusion(circuit, quantum):
    circuit.h(quantum)
    circuit.x(quantum)

    CCCZ(circuit, quantum)

    circuit.x(quantum)
    circuit.h(quantum)

    circuit.barrier(quantum)
```

`Grover` Function

The `Grover` function is defined, combining the steps of initializing the superposition, applying the oracle, and performing diffusion operators iteratively. This function allows Grover's algorithm to search for the marked state efficiently.

```
def Grover(circuit, quantum, target_state):
    circuit.h(quantum)
    oracle(circuit, quantum, target_state)
    diffusion(circuit, quantum)
    diffusion(circuit, quantum)
    diffusion(circuit, quantum)
```

Quantum and Classical Registers

A quantum register with 4 qubits and a classical register with 4 bits are created to store quantum and classical data, respectively.

```
quantum = QuantumRegister(4, "Quantum")
classical = ClassicalRegister(4, "Classical")

circuit = QuantumCircuit(quantum,classical)

for qubit in quantum:
    circuit.reset(qubit)
```

Initialization and Target State

The marked state is set to "1011," and the target state is computed by reversing the marked state. The `Grover` function is then called to construct the quantum circuit, and the circuit is drawn and saved as an image.

```
marked_state = "1011"
target_state = "".join(list(reversed(marked_state))

Grover(circuit, quantum, target_state)

circuit.measure(quantum, classical)
circuit.draw(output='mpl', filename='circuit.png', scale=0.5)
```

Running the Quantum Circuit

The quantum circuit is simulated using the Qiskit Aer simulator. A job is executed with 10024 shots (repetitions), and the result is obtained. Finally, the measurement counts are retrieved, printed, and visualized as a histogram using Qiskit's `plot_histogram` function.

```
simulator = Aer.get_backend('qasm_simulator')
job = execute(circuit, simulator, shots=10024)
result = job.result()

counts = result.get_counts()
print(counts)
plot_histogram(counts)
```

Part 2

Introduction

This section introduces a modification of Grover's algorithm with a double target oracle, allowing the search for two distinct marked states.

Importing Libraries

Necessary libraries are imported for the implementation of the modified algorithm.

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, Aer,
execute
from qiskit.visualization import plot_histogram
from math import pi
```

`double_target_oracle` Function

The `double_target_oracle` function is defined, constructing an oracle with two target states. It applies a phase flip for each of the two target states using the `CCCZ` gate.

```
def double_target_oracle(circuit, quantum, target_state1, target_state2):
    for i, bit in enumerate(target_state1):
        if bit == '0':
            circuit.x(quantum[i])
    CCCZ(circuit, quantum)
    for i, bit in enumerate(target_state1):
        if bit == '0':
            circuit.x(quantum[i])

    for i, bit in enumerate(target_state2):
        if bit == '0':
            circuit.x(quantum[i])
    CCCZ(circuit, quantum)
    for i, bit in enumerate(target_state2):
        if bit == '0':
            circuit.x(quantum[i])
```

`grover_diffusion` Function

This function redefines the `grover_diffusion` function, which creates the Grover diffusion operator, the same as in Part 1.

```
def grover_diffusion(circuit, quantum):
    circuit.h(quantum)
    circuit.x(quantum)
    CCCZ(circuit, quantum)
    circuit.x(quantum)
    circuit.h(quantum)
```

Quantum and Classical Registers

A quantum register with 4 qubits and a classical register with 4 bits are created again for storing quantum and classical data.

```
quantum = QuantumRegister(4, "Quantum")
classical = ClassicalRegister(4, "Classical")
```

Superposition and Target States

Hadamard gates are applied to create a superposition of all states. Two target states, "0010" and "1110," are defined for the double target oracle.

```
circuit = QuantumCircuit(quantum, classical)
circuit.h(quantum)
marked_state1 = "0010"
marked_state2 = "1110"
```

Grover Iterations

The number of Grover iterations is set (currently 1), and Grover's algorithm is executed. The double target oracle and Grover diffusion operator are applied iteratively.

```
num_iterations = 1

for _ in range(num_iterations):
    double_target_oracle(circuit, quantum, target_state1, target_state2)
    grover_diffusion(circuit, quantum)
```

Running the Quantum Circuit

Similar to Part 1, this cell simulates the quantum circuit using the Qiskit Aer simulator, executes the job, retrieves the measurement counts, and visualizes the results as a histogram.

```
job = execute(circuit, simulator, shots=1024)
result = job.result()
counts = result.get_counts()
plot_histogram(counts)
```

The provided code successfully implements Grover's algorithm, allowing efficient search for marked states in an unsorted database.