Table of Contents

Introduction

My project is a Raspberry Pi quantum computer that performs five different quantum programs: Quantum Coin Tosser, True Random Number Generator, Two Qubit Bell State, Three Qubit GHZ State and Quantum Art. To do this I used a software called Qiskit that simulates a basic quantum computer and can also run the programs in a real quantum computer. To clarify, I will not convert the Raspberry Pi into a quantum computer, instead it simulates one.

Furthermore, the programs show simple examples of entanglement and superposition. I used the SenseHat 8x8 LED display to present the results of the quantum programs and the joystick, for the user to choose the programs and interact with the computer.

Definitions, Acronyms and Abbreviations

- Quantum Computing: Uses quantum mechanics to solve problems too complex for classical computers.
- Qubits: a classical processor uses bits to perform its operations. Quantum computers use qubits (quantum bits) to run multidimensional quantum algorithms. The state of a qubit is determined by the spin of an electron.
- Superposition: when a qubit can be in multiple states at the same time.
- Entanglement: essentially, lack of independence.
- IBM Q System One: IBM offers the first commercial, approximate quantum computing system by giving cloud access to quantum computers. With the main hardware, cryostats and quantum chips all in the IBM labs.
- Qiskit: a software to simulate a basic quantum computer. Running a quantum computer emulator on Raspberry Pi. IBM's open source quantum computing software framework

Project Detail

The Raspberry Pi quantum computer is a simulator of a real quantum computer. It can run simple quantum programs. This project is composed of three main elements, as shown in Figure 1. The first one is the Raspberry Pi, which will have a microSD card with a software development kit called Qiskit. The second element is the SenseHAT LED display. This add-on 8x8 LED board is easily connected to the Raspberry Pi, and displays the results of the quantum programs. The circuit diagram of the 8x8 LED grid is shown on Figure 2. The third element of the project is the SenseHAT joystick. The purpose of this element is to interact with the computer and choose the programs, or change the interfaces.

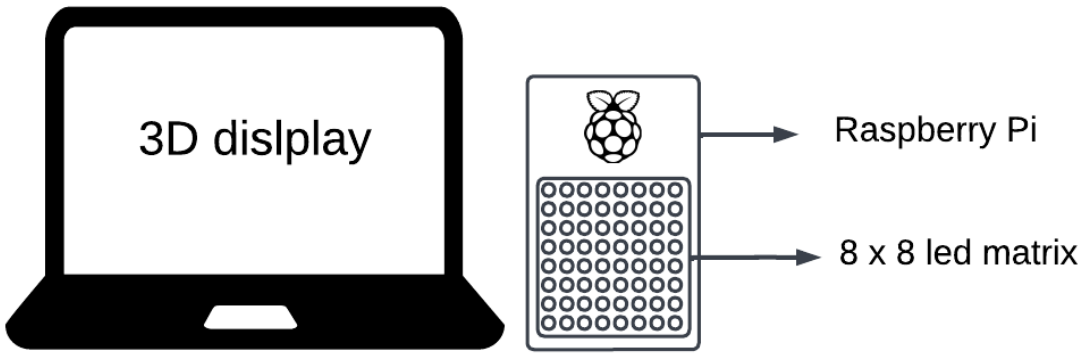

Figure 1. *CAD drawing for whole project*

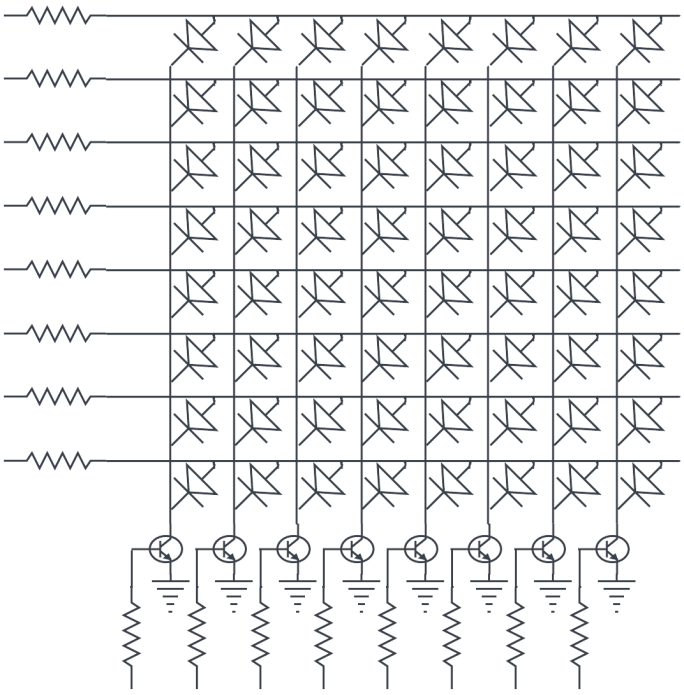

Figure 2. *Circuit diagram for 8x8 LED board*

The software side includes four programs that involve superposition and entanglement. The programs are a Quantum Coin Tosser, True Random Number Generator, Two Qubit Bell State and Three Qubit GHZ State.The results of these programs are displayed on the SenseHAT 8x8 LED matrix.

Quantum Coin Tosser: Coin shows heads and the computer will play first. It can choose to flip the coin or not, but the user doesn't get to see the outcome. Next it's the user's turn. The can choose to flip the coin or not, and their move won't be revealed to the computer. Finally, the computer. Finally the computer plays again, and can flip their coin or not. The coin is revealed and if it's heads, the computer wins, if it's tails, the user. The computer will win 100% of the time. This is because the quantum computer sets the qubit into superposition so no matter if the

user decides to flip or not it doesn't affect the outcome, so when the quantum computer goes again and sets it back to the original heads value.

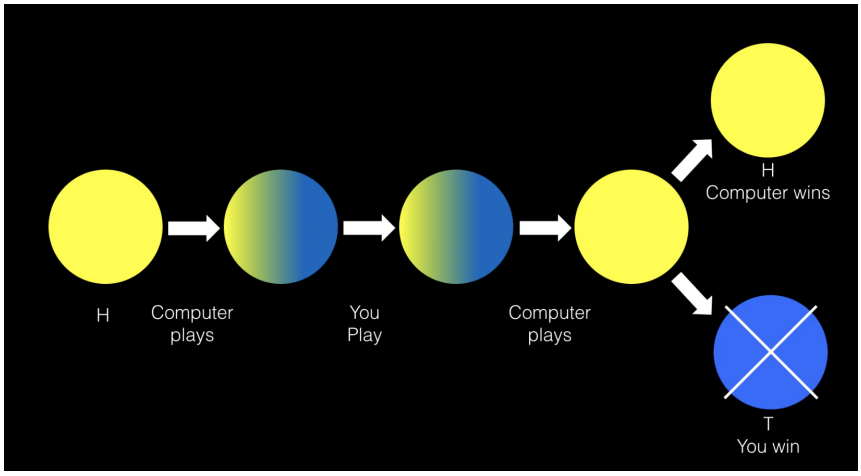

Figure 3. *Quantum Coin Tosser*

True Random Number Generator: Real computers don't have true randomness, instead they are pseudorandom. In this program there are 6 qubits that are in superposition so when measured the numbers generate a randomly generated string of bits. Which is then converted to a decimal number.

Two Qubit Bell State: Maximally entangled quantum state of two qubits, even if spatially separated. If the first qubit is measured, the second qubit will always have the same value as the first.

Three Qubit GHZ State: Entangled quantum state for three qubits. Can only be 000 or 111. Before measured there is a 0.5 possibility for each of those states.

Quantum Art: Mixes the four other programs together. It generates randomly generated art. To do so first a two qubit superposition is measured and it generates 4 different possible numbers. From the number it creates a bell state circuit and GHZ circuit with random entanglements. Then those circuits are measured and generate a random 4 bit sequence. The four

bit sequences are multiplied to create a 2D array, so it displays different colors depending on the number.

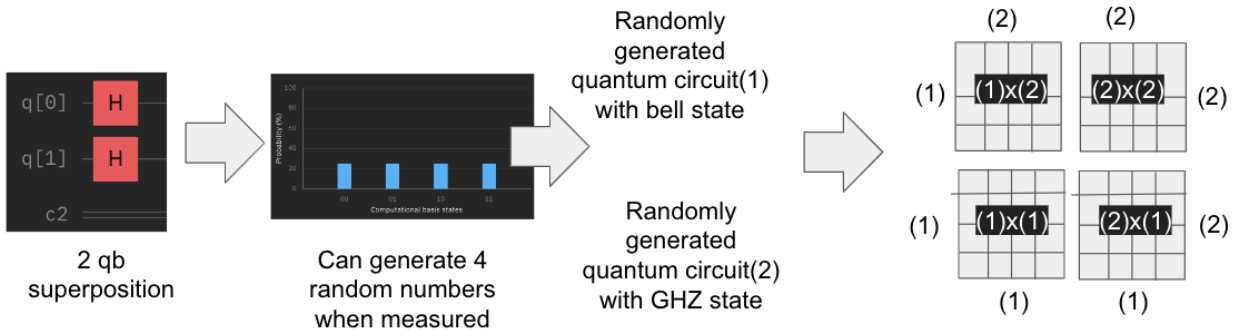

Figure 4. *Quantum Art*

Cost Analysis

The projected cost was almost accurate, it was off by 26 dollars because it was decided not to include a battery, instead the computer runs when it is plugged in.

Projected Cost

Expense	Description	Cost
Wages	Project Manager and Lead Developer will work for 20 hours each. So a total of 40 hours, for $30 an hour	$1,200
RaspberryPi kit	RaspberryPi kit used for CS121 Computer Organization class	$120
SenseHAT LED display	Add-on 8x8 LED board for RaspberryPi	$30
microSD card	16 GB microSD card	$8
Battery pack	Rechargeable USB-C battery pack	$26
TOTAL		$1,384

Table 1. *Projected cost table*

Actual cost

Expense	Description	Cost
Wages	Project Manager and Lead Developer will work for 20 hours each. So a total of 40 hours, for $30 an hour	$1,200
RaspberryPi kit	RaspberryPi kit used for CS121 Computer Organization class	$120
SenseHAT LED display	Add-on 8x8 LED board for RaspberryPi	$30
microSD card	16 GB microSD card	$8
TOTAL		$1,358

Table 1. *Actual cost table*

Project Plan

Role definition

Since I am working as a team of one. There is technically just one position. But to better organization the tasks are divided in two roles:

- Project manager: Does all the documentation, which include proposal/pitch, status updates, progress report, presentation and final report. It also includes managing the hardware, which in this case includes ordering pieces and setting the hardware up for the project to run.
- Lead Developer: Researches about quantum computing, codes all the programs. Also in charge of the implementation of both the 2D led display, and the possibility of a 3D display.

Projected vs Actual Time

The projected timeline was not as accurate as the projected cost. The Project Manager underestimated how long it would take the Lead Developer to learn and research about Quantum Computing. So the project ended up being done around a month after it was predicted, but still in time.

Milestone	Task Ownership	Projected Timeline	Actual Timeline
Order battery and LED display, and buy microSD card	Project manager	Oct 7	Oct 7
Learn and research about quantum computing	Lead Developer	Oct 7 - Oct 26	Oct 7 - Nov 25

Install qiskit on Raspberry pi	Lead Developer	Oct 17	Oct 17
Learn and code programs in quiskit	Lead Developer	Oct 18 - Nov 1	Nov 14 - Nov 18
Research and practice about manipulating led grid to show 2D results	Lead Developer	Nov 1 - Nov 7	Nov 21 - Nov 25
Research about 3D display and implement (if possible)	Lead Developer	Nov 1 - Nov 10	Nov 28 - Dec 2
Run the programs and coordinate with displays - Project manager	Project manager	Oct 17 - Nov 11	Nov 14 - Dec 5

Table 2. *Milestones table*

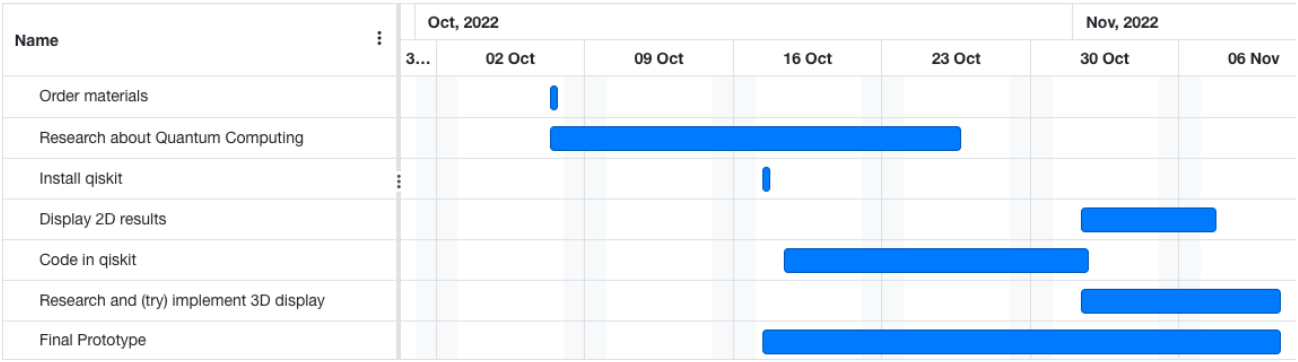

Figure 3. *Projected Gantt chart*

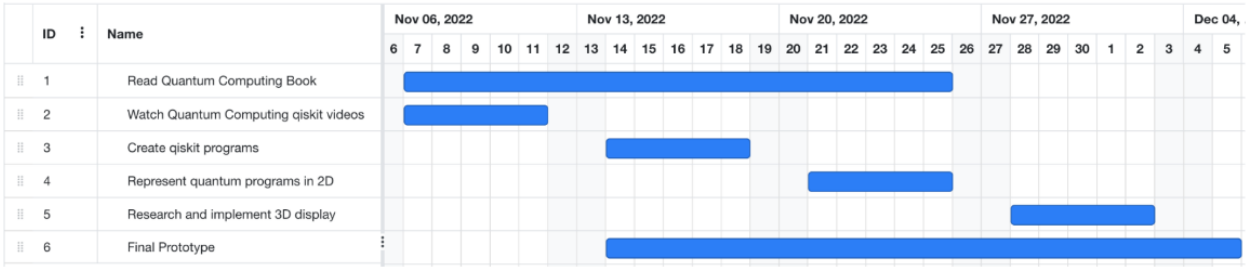

Figure 4. *Actual Gantt chart*

Target Market

Quantum computing is one of the most innovative fields in technology. Therefore, the Quantum Computing market is expected to grow from 472 million dollars in 2021 to 1,765 million by 2026. The target market for my project would be anyone interested in learning about quantum computing, considering the market is growing so fast, the project would be increasingly marketable in the upcoming years.

Reflection

Overall, the quantum computer was a project I enjoyed doing. It was interesting learning the basics about quantum mechanics and computing. The most challenging part of the project was time management. Also installing packages to the RaspberryPi was a lot more complicated and time consuming than I thought. My original plan was to make a headless RaspberryPi, so it wouldn't have to be connected to my laptop to run, but I was unable to get my jupyter notebook to work with it. Also I ended up not having enough time to implement the 3D display. My take-aways are to always have a backup plan, be versatile and flexible because most of the time things don't work the way you think they will.

Code

main_controller.py

```
"""
   Main controller for Qiskit on Raspberry PI SenseHat
"""
from sense_hat import SenseHat
from urllib.request import urlopen
import time
from qiskit import IBMQ, execute # qiskit class
from qiskit import BasicAer as Aer # qiskit class
import Qconfig_IBMQ_experience  as config
from qiskit.providers.ibmq import least_busy # qiskit class
import random_num
import coin_tosser
import bell
import ghz

sense = SenseHat()

# Background icon
X = [255, 0, 255]  # Magenta
Y = [255,192,203] # Pink
P = [255,255,0] #Yellow
O = [0, 0, 0]  # Black
B = [0,0,255] # Blue
W = [255, 255, 255] #White

# Check if online
def internet_on():
   try:
       response = urlopen('https://www.google.com', timeout=10)
       return True
   except:
       return False

print("Getting provider...")
sense.show_message("Getting provider...", text_colour=W)

if not IBMQ.active_account():
```

```
    if internet_on():
        IBMQ.save_account(config.APItoken)
        IBMQ.load_account()
        provider = IBMQ.get_provider()
    else:
        sense.show_message("Offline mode",text_colour= W)

# Set default SenseHat configuration.
sense.clear()
sense.low_light = True

IBM_Q_B = [
B, B, B, W, W, B, B, B,
B, B, W, B, B, W, B, B,
B, W, B, B, B, B, W, B,
B, W, B, B, B, B, W, B,
B, W, B, B, B, B, W, B,
B, P, W, B, B, W, B, B,
P, P, P, W, W, B, B, B,
B, P, B, W, W, W, B, B
]

IBM_AER = [
O, O, W, W, W, W, O, O,
O, W, W, O, O, W, W, O,
W, W, W, O, O, W, W, W,
W, W, O, W, W, O, W, W,
W, W, O, O, O, O, W, W,
W, O, W, W, W, W, O, W,
O, W, O, O, O, O, W, O,
O, O, W, W, W, W, O, O
]


# Function to set the backend
def set_backend(back):
    global backend
    if back == "ibmq" and internet_on():
        sense.show_message("Getting best backend...", text_colour=W)
        backend = least_busy(provider.backends(n_qubits=5, operational=True,
simulator=False))
```

```
        sense.show_message(backend.name(), text_colour=W)
        status = backend.status()
        is_operational = status.operational
        jobs_in_queue = status.pending_jobs
        sense.show_message(str(jobs_in_queue), text_colour=W)
        sense.set_pixels(IBM_Q_B)
    else:
        backend = Aer.get_backend('qasm_simulator')
        sense.show_message(backend.name(), text_colour=W)
        sense.set_pixels(IBM_AER)
    print(backend.name)

# Load the Qiskit function files. Showing messages when starting and when done.
sense.show_message("Qiskit", text_colour=W)

# Initialize the backend to AER
back = "aer"
set_backend(back)


# The main loop.
# Use the joystick to select and execute one of the Qiskit function files.

while True:
    joy_event = sense.stick.get_events()
    if len(joy_event) > 0 and joy_event[0][2]=="pressed":
        # 2 qubit bell state (entanglement)
        if joy_event[0][1]=="up":
            sense.show_message("Bell", text_colour=B)
            bell.bell_execute(backend,back)
        else:
            # 3 qubit GHZ state  (entanglement)
            if joy_event[0][1]=="down":
                sense.show_message("GHZ", text_colour=B)
                ghz.ghz_execute(backend,back)
            else:
                # Coin tosser
                if joy_event[0][1]=="left":
                    sense.show_message("Coin", text_colour=B)
                    coin_tosser.coin_execute(backend,back)
                else:
```

```
                    # Random number generator
                    if joy_event[0][1]=="right":
                        sense.show_message("Random", text_colour=B)
                        random_num.rand_execute(backend,back)

                    else:
                        # Choose art
                        if joy_event[0][1]=="middle":
                            if back == "aer":
                                back = "ibmq"
                                set_backend(back)
                            else:
                                back = "aer"
                                set_backend(back)
```

histogram_maker.py

```
"""
   Using SenseHat 8x8 display to show bar graph of 2 or 3 qubit Qiskit results
dictionaries
"""
# Start by importing and simplifying required modules.
from sense_hat import SenseHat
#from sense_emu import SenseHat
hat = SenseHat()


# Defining the SenseDisplay function.
def SenseDisplay(InputDict,Qbits,back):
   # Create a default Qdict dictionary with all values 0
   global lst
   lst = [bin(x)[2:].rjust(Qbits, '0') for x in range(2**Qbits)]
   values = [0]*pow(2,Qbits)
   Qdict = dict(zip(lst,values))

   # Update the dictionary with the actual dictionary values sent to the function.
   Qdict.update(InputDict)
```

```
   # Scale by dividing by 1024 (shots) - For now assuming 1024, which is set by the sh
parameter.
   # Defining the display colors.
   red = (255, 0, 0)
   green = (0, 255, 0)
   blue = (0, 0, 255)

   hat.clear()
   hat.set_rotation(270)

   # Writing to the SenseHat display pixels.
   for key in Qdict:
       y=7-int(key,2) # Cycle through the states
       for x in range (0,8): # Cycle through the pixels
               val = ((x+1)*128)-Qdict[key] # The difference between the state result
and the pixel x position
               if val<0:
                       #If the state result is greater than the pixel, set pixel color
red.
                       color=red
               else:
                   if val>0 and val<128:
                       #If the state result is within the pixel, set pixel color
gradient.
                       fade = (255-(2*val),0,(2*val))
                       color=fade
                   else:
                       #If the state result is less than the pixel, set pixel color
blue.
                       color=blue
               #Set pixel color.
               hat.set_pixel(x, y, color)
```

Hassi Norlen (2021) Grasp (V0.6) [Python].

https://github.com/ordmoj/qrasp

art.py

```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute, IBMQ,
Aer
from sense_hat import SenseHat
hat = SenseHat()

# Colors
blue = (0, 0, 255)
white = (255,255,255)
nothing = (0,0,0)
pink = (255,105, 180)
purple = (255, 0, 255)
light_blue = (173, 216, 230)
red = (255, 0, 0)
green = (0, 255, 0)

# Define the execute function that is called by the main controller.
def rand_num(backend, back):
   n = 2
   sh = 1
   # Create a Quantum Circuit with 2 qubits and 2 bits
   q = QuantumRegister(n,'q')
   c = ClassicalRegister(n,'c')
   circuit = QuantumCircuit(q,c)
   circuit.h(q) # Applies hadamard gate to all qubits
   circuit.measure(q,c) # Measures all qubits
   # Create a Quantum Program for execution of the circuit on the selected backend
   job = execute(circuit, backend, shots=1)
   # Get the result of the execution and convert to decimal
   result = job.result()
   counts = result.get_counts(circuit)
   binary = list(counts.keys())
   decimal = int(binary[0],2)
   return decimal

def art_execute(backend, back):
   # Create circuit with bell state
   num = rand_num(backend, back)

   n = 4
```

```
q = QuantumRegister(n, 'q')
c = ClassicalRegister(n, 'c')
circuit1 = QuantumCircuit(q, c)

circuit1.h(q[0])
circuit1.h(q[1])
circuit1.h(q[2])
circuit1.h(q[3])



if num == 0:
    circuit1.cx(q[0], q[2])
elif num == 1:
    circuit1.cx(q[1], q[3])
elif num == 2:
    circuit1.cx(q[2], q[3])
elif num == 3:
    circuit1.cx(q[0], q[3])

circuit1.measure(q,c) # Measures all qubits
# Create a Quantum Program for execution of the circuit on the selected backend
job = execute(circuit1, backend, shots=1)
# Get the result of the execution and convert to decimal
result = job.result()
counts = result.get_counts(circuit1)
binary = list(counts.keys())
print(binary)

# Create circuit with GHZ state
rand_num2 = rand_num(backend, back)
n = 4

# Create circuit
q = QuantumRegister(n, 'q')
c = ClassicalRegister(n, 'c')
circuit2 = QuantumCircuit(q, c)

circuit2.h(q[0])
circuit2.h(q[1])
circuit2.h(q[2])
```

```
circuit2.h(q[3])



if rand_num2 == 0:
    circuit2.cx(q[0], q[2])
    circuit2.cx(q[0], q[3])
elif rand_num2 == 1:
    circuit2.cx(q[1], q[3])
    circuit2.cx(q[1], q[2])
elif rand_num2 == 2:
    circuit2.cx(q[2], q[3])
    circuit2.cx(q[2], q[0])
elif rand_num2 == 3:
    circuit2.cx(q[0], q[3])
    circuit2.cx(q[0], q[1])

circuit2.measure(q,c) # Measures all qubits
# Create a Quantum Program for execution of the circuit on the selected backend
job = execute(circuit2, backend, shots=1)
# Get the result of the execution and convert to decimal
result = job.result()
counts = result.get_counts(circuit2)
binary2 = list(counts.keys())

# 1*1 square
rows, cols = (4, 4)
bin = str(binary)[2:-2]
arr = [[0 for i in range(cols)] for j in range(rows)]
for i in range(4):
    for j in range(4):
        arr[i][j] = bin[i] + bin[j]
        if arr[i][j] == '00':
            hat.set_pixel(i,j+4, green)
        elif arr[i][j] == '01':
            hat.set_pixel(i,j+4, pink)
        elif arr[i][j] == '10':
            hat.set_pixel(i,j+4, red)
        elif arr[i][j] == '11':
            hat.set_pixel(i,j+4, blue)

# 1*2 square
```

```
rows, cols = (4, 4)
bin = str(binary)[2:-2]
bin2 = str(binary2)[2:-2]
arr = [[0 for i in range(cols)] for j in range(rows)]
for i in range(4):
    for j in range(4):
        arr[i][j] = bin[i] + bin2[j]
        if arr[i][j] == '00':
            hat.set_pixel(i+4,j+4, green)
        elif arr[i][j] == '01':
            hat.set_pixel(i+4,j+4, pink)
        elif arr[i][j] == '10':
            hat.set_pixel(i+4,j+4, red)
        elif arr[i][j] == '11':
            hat.set_pixel(i+4,j+4, blue)

# 2*1 square
rows, cols = (4, 4)
bin = str(binary)[2:-2]
bin2 = str(binary2)[2:-2]
arr = [[0 for i in range(cols)] for j in range(rows)]
for i in range(4):
    for j in range(4):
        arr[i][j] = bin2[i] + bin[j]
        if arr[i][j] == '00':
            hat.set_pixel(i,j, green)
        elif arr[i][j] == '01':
            hat.set_pixel(i,j, pink)
        elif arr[i][j] == '10':
            hat.set_pixel(i,j, red)
        elif arr[i][j] == '11':
            hat.set_pixel(i,j, blue)

# 2*2 square

rows, cols = (4, 4)
bin = str(binary)[2:-2]
bin2 = str(binary2)[2:-2]
arr = [[0 for i in range(cols)] for j in range(rows)]
for i in range(4):
    for j in range(4):
```

```
            arr[i][j] = bin2[i] + bin2[j]
            if arr[i][j] == '00':
                hat.set_pixel(i+4,j, green)
            elif arr[i][j] == '01':
                hat.set_pixel(i+4,j, pink)
            elif arr[i][j] == '10':
                hat.set_pixel(i+4,j, red)
            elif arr[i][j] == '11':
                hat.set_pixel(i+4,j, blue)
```

bell.py

```
"""
   Two qubit quantum cirquit and sets up and measures a Bell, or entangled, state.
"""
# Start by importing and simplifying required modules.
from sense_hat import SenseHat
hat = SenseHat()

# Define the execute function that is called by the main controller.
def bell_execute(backend,back):
   from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
   from qiskit import execute
   import numpy as np
   #Set number of bits and number of shots
   n = 2
   print(n)
   sh = 1024
   # Create circuit
   qr = QuantumRegister(n)
   cr = ClassicalRegister(n)
   circuit = QuantumCircuit(qr, cr)

   # Add gates to the circuit
   circuit.h(qr[0])
   circuit.cx(qr[0], qr[1])
   circuit.measure(qr[0], cr[0])
```

```
   circuit.measure(qr[1], cr[1])
    job = execute(circuit, backend, shots=sh)
   # Get the result of the execution
   result = job.result()
   # Provide the results
   print ("Results:")
   #print (result)
   Qdictres = result.get_counts(circuit)
   print(Qdictres)

   # Import the Raspberry PI SenseHat display function.
   from histogram_maker import SenseDisplay
   # Display the quantum dictionary as a bar graph on the SenseHat 8x8 pixel display
by calling the SenseDisplay function.
   SenseDisplay(Qdictres,n,back)
```

coin_tosser.py

```
# Python function that creates a quantum coin tosser
from sense_hat import SenseHat
from qiskit import QuantumCircuit, Aer, IBMQ, QuantumRegister, ClassicalRegister,
execute

hat = SenseHat()

X = [255, 0, 255]  # Magenta
R = [255, 0, 0] # Red
P = [255,255,0] #Yellow
O = [0, 0, 0]  # Black
B = [0,0,255] # Blue
W = [255, 255, 255] #White
G = [0, 255, 0] # Green

# Building the initial circuit
def initial_circuit():
   circuit = QuantumRegister(1, 'circuit')
   measure = ClassicalRegister(1, 'result')
   qc = QuantumCircuit(circuit, measure)
   return qc, circuit, measure

def coin_execute(backend,back):
```

```
qc, circuit, measure = initial_circuit()

# First turn
hat.show_message("QC", text_colour=X)
qc.h(circuit[0])

# Second turn
hat.show_message("You", text_colour=X)

run_program = True
while run_program == True:
    joy_event = hat.stick.wait_for_event()
    if len(joy_event) > 0 and joy_event.action=="pressed":
        if joy_event.direction=="up":
            # Flip coin
            hat.show_message("Flip", text_colour=W)
            run_program = False
        else:
            # No flip
            if joy_event.direction=="down":
                hat.show_message("No flip", text_colour=X)
                run_program = False
            else:
                if joy_event.direction=="left":
                    hat.show_message("No flip", text_colour=X)
                    run_program = False
                else:
                    if joy_event.direction=="right":
                        hat.show_message("No flip", text_colour=X)
                        run_program = False
                    else:
                        if joy_event.direction=="middle":
                            hat.show_message("No flip", text_colour=X)
                            run_program = False




# Third turn
hat.show_message("QC", text_colour=X)
qc.h(circuit[0]) # Remove superposition
```

```
    qc.measure(circuit, measure)

    job = execute(qc, backend, shots=8192)
    res = job.result().get_counts()

    # Winner
    if len(res) == 1 and list(res.keys())[0] == '0':
        hat.show_message("QC Wins",text_colour=R)
    if len(res) == 1 and list(res.keys())[0] == '1':
        hat.show_message("You Win", text_colour=G)
    if len(res) == 2:
        hat.show_message("Tie", text_colour=X)
```

ghz.py

```
"""
    Creates a simple, three qubit quantum cirquit and sets up and measures an entangled
GHZ state.
    Displays a histogram
"""

from sense_hat import SenseHat
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import execute

hat = SenseHat()

# Define the execute function that is called by the main controller.
def ghz_execute(backend,back):

    #Set number of bits and number of shots
    n = 3
    sh = 1024

    # Create circuit
    qr = QuantumRegister(n)
    cr = ClassicalRegister(n)
    circuit = QuantumCircuit(qr, cr)
```

```
   # Add gates to the circuit
   circuit.reset(qr[0])
   circuit.reset(qr[1])
   circuit.reset(qr[2])
   circuit.h(qr[0])
   circuit.cx(qr[0], qr[1])
   circuit.cx(qr[0], qr[2])
   circuit.measure(qr[0], cr[0])
   circuit.measure(qr[1], cr[1])
   circuit.measure(qr[2], cr[2])

   job = execute(circuit, backend, shots=sh)
   # Get the result of the execution
   result = job.result()
   # Provide the results
   print ("Results:")
   #print (result)
   Qdictres = result.get_counts(circuit)
   print(Qdictres)

   # Import the Raspberry PI SenseHat display function.
   from histogram_maker import SenseDisplay
   # Display the quantum dictionary as a bar graph on the SenseHat 8x8 pixel display
by calling the SenseDisplay function.
   SenseDisplay(Qdictres,n,back)
```

random_num.py

```
# Python function that creates a a true random number generator

from sense_hat import SenseHat
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute
import numpy as np

hat = SenseHat()

X = [255, 0, 255]  # Magenta
```

```
# Define the execute function that is called by the main controller.
def rand_execute(backend,back):
   n = 6
   sh = 1
   # Create a Quantum Circuit with 16 qubits and 16 bits
   q = QuantumRegister(n,'q')
   c = ClassicalRegister(n,'c')
   circuit = QuantumCircuit(q,c)
   circuit.h(q) # Applies hadamard gate to all qubits
   circuit.measure(q,c) # Measures all qubits

   # Create a Quantum Program for execution of the circuit on the selected backend
   job = execute(circuit, backend, shots=1)
   # Get the result of the execution and convert to decimal
   result = job.result()
   counts = result.get_counts(circuit)
   binary = list(counts.keys())
   decimal = int(binary[0],2)

   hat.show_message(str(decimal), text_colour=X)
```

References

(2021). (rep.). *Quantum Computing Market by Offering (Systems and Services), Deployment (On Premises and Cloud Based), Application, Technology, End-use Industry and Region (2021-2026)*. Retrieved from https://www.marketsandmarkets.com/Market-Reports/quantum-computing-market-144888301.html.

Bate, A. (2021, September 18). *IBM Q System one quantum computing on a Raspberry Pi?* Raspberry Pi. Retrieved October 6, 2022, from https://www.raspberrypi.com/news/ibm-q-system-one-quantum-computing-raspberry-pi/

Coward, C. (2022, October 5). *Turn your raspberry pi into a quantum computer with Qrasp*. Hackster.io. Retrieved from https://www.hackster.io/news/turn-your-raspberry-pi-into-a-quantum-computer-with-qrasp-4ccde390a48a

Norlen, H. (2020, July 22). *QRASP-quantum on a Raspberry Pi*. Medium. Retrieved from https://medium.com/qiskit/qrasp-a-wee-quantum-computer-74ef7f927b1e

Vandeth, D., Harishankar, R., Holt, B., & Chow, P. G. and J. (2021, February 9). *Quantum computing*. IBM Research. Retrieved from https://research.ibm.com/quantum-computing

What is quantum computing? IBM. (n.d.). Retrieved from https://www.ibm.com/topics/quantum-computing

Wilczek, F., & Quanta Magazine moderates comments to facilitate an informed, substantive. (2019, May 23). *Entanglement made simple*. Quanta Magazine. Retrieved from https://www.quantamagazine.org/entanglement-made-simple-20160428/