ECG Research

Mentor: Professor Cai

Content Page

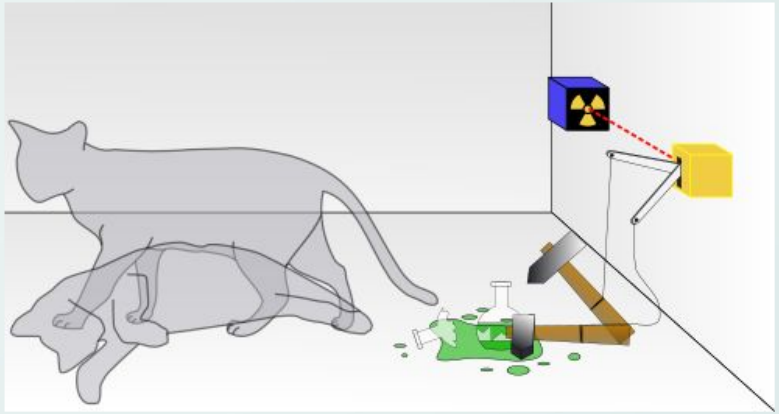
Quantum Research

Readups on things related to QM

1) Schrodinger's Cat

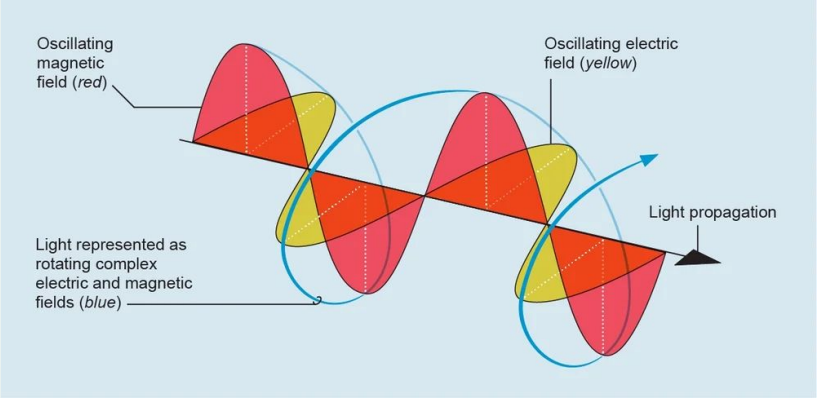
The Schrödinger's cat **demonstrates** the behaviour of quantum particles (in a state of superposition), that it can be in **multiple** states/places at once, only collapsing into a fixed state/place upon **observation**

This thought experiment gives rise to various other interpretations, which I will cover later



2) Complex Series

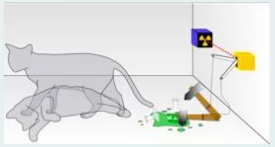
When dealing with quantum mechanics, **predictions** to a behaviour will be needed to be made. However, just by pure real numbers, this prediction will oftentimes not work. Furthermore, it is also useful in **graphical representations** of quantum particles as it allows both the amplitude and phase information to be stored

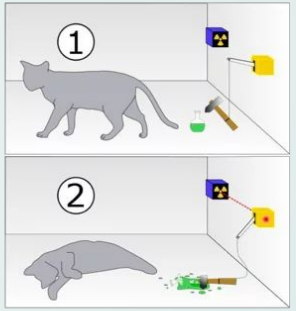


The above image is just one example of its application. For a light wave, having complex will allow it to be displayed 3 dimensionally and hence able to understand it's movement and behaviours better

3) Different Interpretations - Copenhagen Interpretation

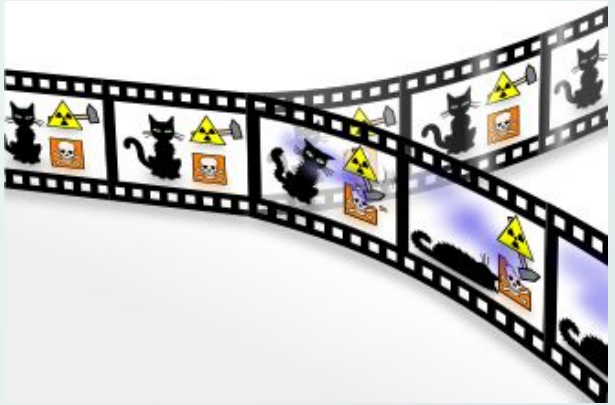
Copenhagen Interpretation tells us that quantum particles are in a state of superposition until being observed, which causes the superpositional wave collapses down into one single particle/state and is observed by us.





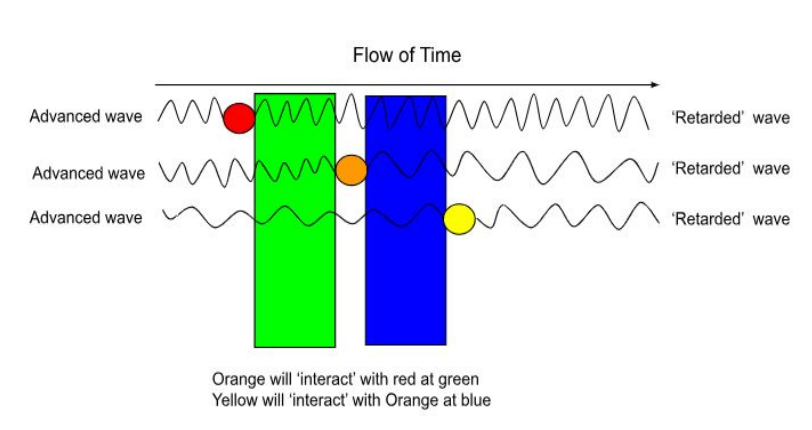
3) Different Interpretations - Many World Interpretation

Many World Interpretation suggests that all the possibilities exists, just that they exist in different universes, all connected by a huge and complex wave function. We are only entangled to one part of this complex wave, hence we can only observe one outcome. All of the other outcomes will exist in a separate universe



3) Different Interpretations - Transactional Interpretation

Transactional Interpretation suggests that the **future** particles will **affect the outcome** in **past**. This is because every particle will admit 2 waves, one retarded wave going forward in time and one advanced wave going backwards. They will interact with each other, causing the future to affect the present



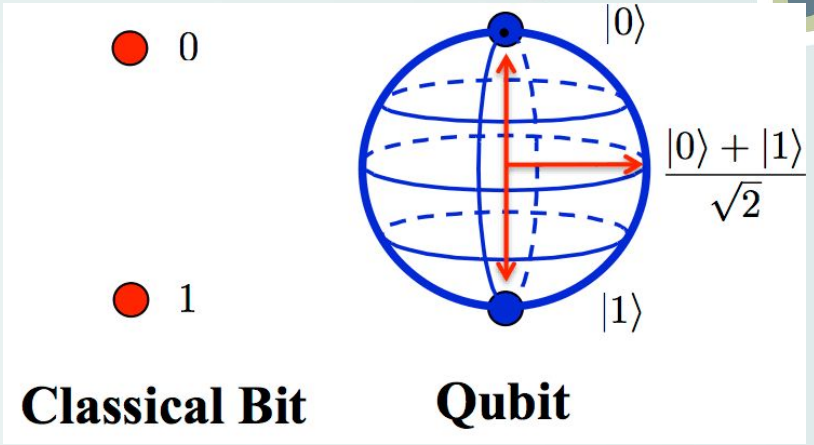
3) Different Interpretations - Ensemble Interpretation

Ensemble Interpretation suggests that everything will affect every other similar things simultaneously, and the results are based on other similar experiments that was done in the past, both consciously and subconsciously. Hence, there is no state of superposition since everything is being determined by other similar experiments

Part 2:
Simulation using Qiskit

4) Qubit

Qubits are essentially bits but used for quantum computing instead of classical computing. It has more processing power and is faster at running the algorithms as compared to classical bits due to its ability to display superposition. However, currently, it takes too much energy to ensure qubits run effectively, hence it is not being widely used



The picture below shows the difference between a classical bit and a qubit. Unlike classical bit which shows two fixed states, whereas for a qubit it has a probability cloud of the possible states.

5) Quantum Logic Gates

Quantum logic gates are the main foundation of quantum computing, able to **manipulate qubits** in order to process data. The ones that I focused on are the **Pauli XYZ gates, Hadamard Gate and CNOT Gates**

Operator	Gate(s)	Matrix
Pauli-X (X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y (Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z (Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase (S, P)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ (T)		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
Controlled Not (CNOT, CX)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

6) 2D Simulation- Schrodinger's Cat

```
#QuantumRegister is used to create a quantum register with n qubits,
#ClassicalRegister is used to create a classical register with n bits,
#QuantumCircuit is used to create a quantum circuit with qubits and bi
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

#radiioactive particle = rp, quantum circuit = qc
rp = QuantumRegister(1, name="RP")
qc = QuantumCircuit(rp)

#Here, we will implement a Hadamard gate on the radioactive particle.
#This will put the particle in a superposition state, where it can be
#either the |0⟩ or |1⟩ state with 50% probability each.
qc.h(rp[0])

qc.draw("mpl")
```

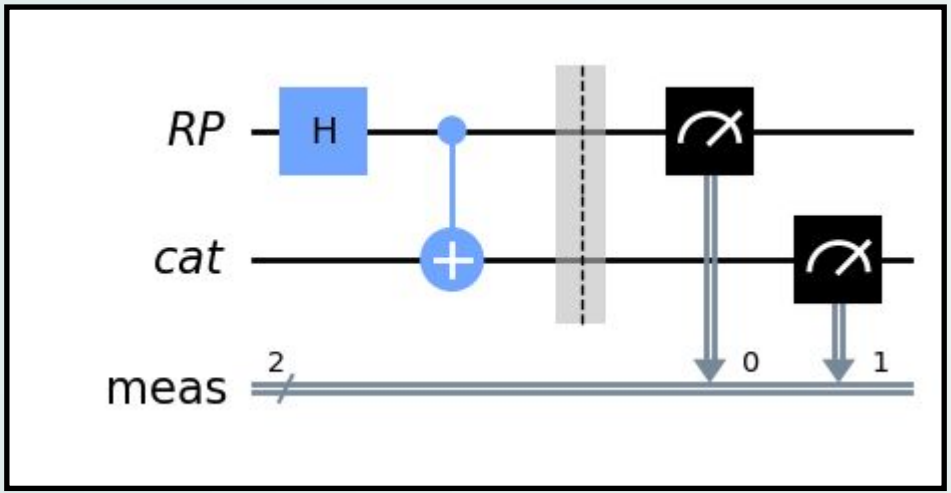
```
#Next, we will implement a classical bit in order to capture the
#measurement of the radioactive particles.
qc.measure_all()

from qiskit_aer import AerSimulator

#Here, we will use the AerSimulator to simulate the quantum circ
#The default number of simulation for qiskit is 1024.
backend = AerSimulator()
result = backend.run(qc).result()

#The result of the simulation is stored in the counts variable,
#and displayed as a dictionary.
#At the same time, this will test whether the gate is truely wor
print(result.get_counts())
```

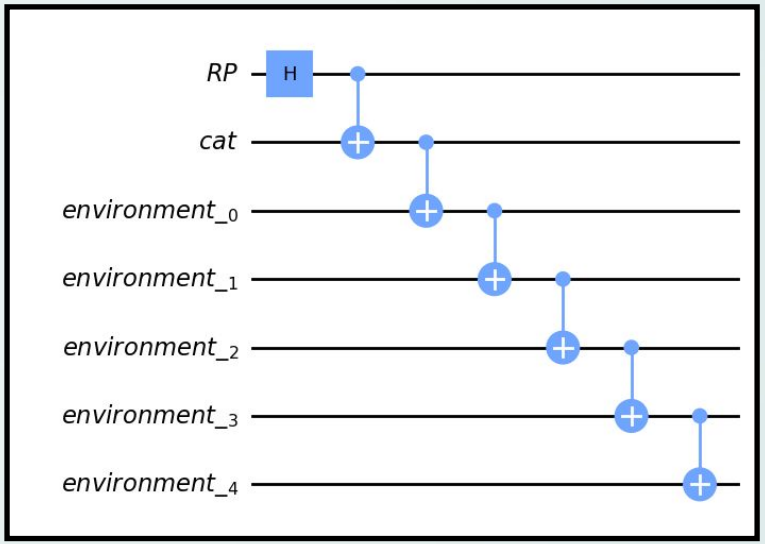
```
cat = QuantumRegister(1, name="cat")
qc = QuantumCircuit(rp, cat)

#similar to before, a Hadamard gate is on the radioactive particle.
qc.h(rp[0])

#Next, we will implement a controlled-NOT gate, where the radioactive
#particle is the control qubit, and the cat is the target qubit.
#This will entangle the radioactive particle and the cat.
#The initial state of the cat is |0⟩, and after the entanglement,
#if the radioactive particle is|1⟩, NOT(cat), causing the cat to be
#in the |1⟩ state, else it remains in the |0⟩ state.
qc.cx(rp[0], cat[0])

qc.draw("mpl")
```

```
#This is similar to the above section
qc.measure_all()

result = backend.run(qc).result()

print(result.get_counts())
```

```
env = QuantumRegister(5, name='environment_')

qc = QuantumCircuit(rp, cat, env)

qc.h(rp)

qc.cx(rp, cat)


qc.cx(cat, env[0])
qc.cx(env[0], env[1])
qc.cx(env[1], env[2])
qc.cx(env[2], env[3])
qc.cx(env[3], env[4])

qc.draw("mpl")
```

6) 2D Simulation- Schrodinger's Cat





6) 2D Simulation- Schrodinger's Cat

$\|00\rangle$	$\|11\rangle$	Total
508	516	1024
520	504	1024
527	497	1024
507	517	1024

Average $|0\rangle$ count: $\frac{508+520+527+507}{4} \approx 515$

Average $|1\rangle$ count: $\frac{509+489+531+508}{4} \approx 509$

% of $|0\rangle$: $\frac{515}{1024} \times 100\% \approx 50.3\%$

% of $|1\rangle$: $\frac{509}{1024} \times 100\% \approx 49.7\%$

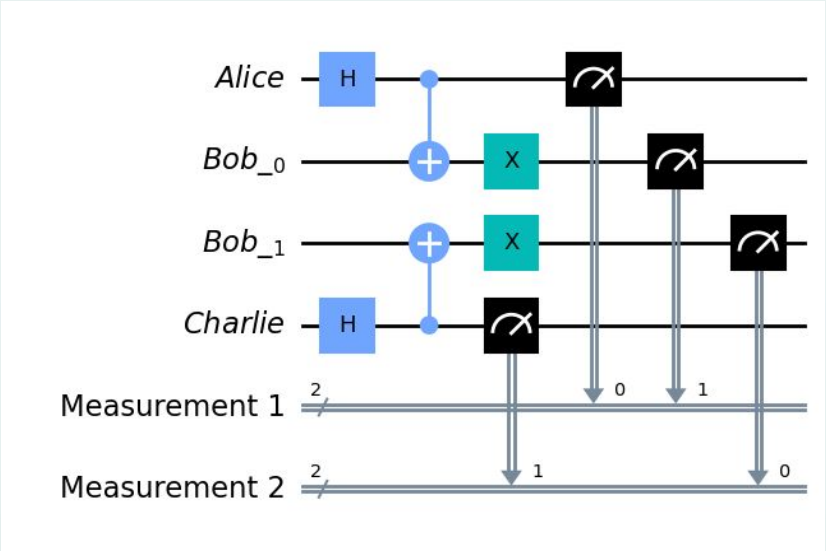
Note: this will get closer to 50% each if ran more times

6) 2D Simulation-
Alice Bob Charlie Experiment

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, Aer, transpile, assemble, execute

# Create named quantum registers for Alice, Bob, and Charlie
alice = QuantumRegister(1, "Alice")
bobs = QuantumRegister(2, "Bob_")
charlie = QuantumRegister(1, "Charlie")

# Create a classical register for measurement outcomes

cr1 = ClassicalRegister(2, "Measurement 1")
cr2 = ClassicalRegister(2, "Measurement 2")

# Create a quantum circuit with three qubits for Alice, Bob, and Charlie
qc = QuantumCircuit(alice, bobs, charlie, cr1, cr2)
```

```
# Create a quantum circuit with three qubits for Alice, Bob, and Charlie
qc = QuantumCircuit(alice, bobs, charlie, cr1, cr2)

# Make Alice and Charlie superposition
qc.h(alice)
qc.h(charlie)

# Entangle Alice and Bob
qc.cx(alice, bobs[0])

# Entangle Bob and Charlie
qc.cx(charlie, bobs[1])

#Ensuring that the measurements of bob is the opposite of alice and charlie
qc.x(bobs[0])
qc.x(bobs[1])
```

6) 2D Simulation-
Alice Bob Charlie Experiment



6) 2D Simulation-
Alice Bob Charlie Experiment

01 01	01 10	10 10	10 01	Total
238	262	259	265	1024
257	251	245	271	1024
272	252	250	250	1024
224	257	272	271	1024

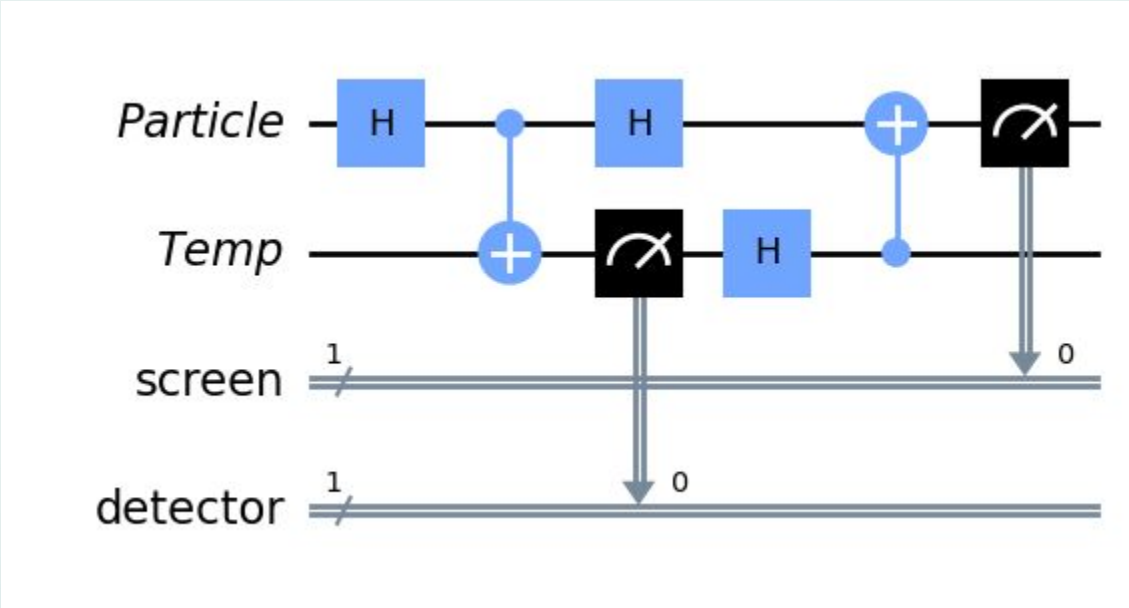
These measurements will show that the 4 possible measurements all have a 25% chance of getting picked, since we assume that Alice and Charlie have no biases over which direction of polarisation they pick. Of course, this is a simplification of the actual experiment, so no imaginary numbers are seen.

6) 2D Simulation-
Delayed Double Slit Experiment

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, Aer, transpile, assemble

# Create quantum registers and classical registers
par = QuantumRegister(1, 'Particle')  # Particle qubit
temp = QuantumRegister(1, 'Temp')      # Temporary qubit
scr = ClassicalRegister(1, 'screen')   # Classical register for screen measurement
det = ClassicalRegister(1, 'detector') # Classical register for detector measurement

# Create a quantum circuit
qc = QuantumCircuit(par, temp, scr, det)
det_present = True
# If the detector is not present, measure Particle (P) directly
if not det_present:
    qc.measure(par, scr)

# If the detector is present, manipulation is needed
else:
    qc.h(par)
    qc.cx(par, temp)
    qc.h(par)
    qc.measure(temp, det)
    qc.h(temp)
    qc.cx(temp, par)
    qc.measure(par, scr)
```

```
backend = Aer.get_backend('qasm_simulator')

# Compile the circuit
qc_compiled = transpile(qc, backend)

# Run the simulation
result = backend.run(assemble(qc_compiled)).result()

# Display the results
counts = result.get_counts(qc)
if det_present:
    print('P T number')
else:
    print('P number')
for keys, values in counts.items():
    print('{} {}'.format(keys, values))
print("Measurement results:", counts)
```

6) 2D Simulation-
Delayed Double Slit Experiment



6) 2D Simulation-
Delayed Double Slit Experiment

10/11	01/00	Total
514	510	1024
502	522	1024
537	507	1024
521	523	1024

Where the photon particle state is represented as the first number of the pair.

Part 3:
With Unity

Short Clip

https://drive.google.com/file/d/1uBCqX7hoSUbXU7wpB9PXdzXLMrWKhlIX/view?usp=drive_link

Codes: Wave Motion

```
using UnityEngine;

2 references
public class WaveGenerator : MonoBehaviour
{
    1 reference
    public float forwardSpeed = 2.0f; // Forward movement speed
    1 reference
    public float waveAmplitude = 1.0f; // Amplitude of the transverse wave
    1 reference
    public float waveFrequency = 1.0f; // Frequency of the transverse wave
    1 reference
    public float waveSpeed = 2.0f; // Speed of the transverse wave

    2 references
    private float originalY;

    0 references
    void Start()
    {
        originalY = transform.position.y;
    }
```

```
void Update()
{
    if (!GetComponent<InputMovement>().IsObserved())
    {
        MoveForward();
        ApplyTransverseWave();
    }
}

1 reference
void MoveForward()
{
    // Move the sphere forward along its local z-axis
    transform.Translate(Vector3.forward * forwardSpeed * Time.deltaTime);
}

1 reference
void ApplyTransverseWave()

// Calculate transverse wave motion along the xz-plane
float waveMotion = waveAmplitude * Mathf.Sin(Time.time * waveFrequency);

// Calculate the target position with transverse wave motion
Vector3 targetPosition = new Vector3(transform.position.x, originalY + waveMotion, transform.position.z);
```

```
// Calculate the interpolation factor based on the smoothness you desire
float smoothness = 0.5f; // Adjust this value to control smoothness
float interpolation = Mathf.Lerp(transform.position.y, targetPosition.y, smoothness * Time.deltaTime);

// Move the particles forward along their local z-axis with smooth transverse wave motion
transform.position = new Vector3(transform.position.x, interpolation, transform.position.z);

// Move the particles forward along their local z-axis with transverse wave motion
float modifiedSpeed = waveSpeed * (1.0f + waveMotion);
transform.Translate(Vector3.forward * modifiedSpeed * Time.deltaTime);


0 references
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("SwitchTrigger"))
    {
        InputMovement inputMovement = GetComponent<InputMovement>();
        // Toggle isObserved boolean using the ToggleObserved method from InputMovement
        inputMovement.ToggleObserved();

        // Check if this is the first WaveGenerator object that hit the switch
        if (inputMovement.IsObserved())
        {
            // Get all WaveGenerator objects in the scene
            WaveGenerator[] waveGenerators = FindObjectsOfType<WaveGenerator>();
```

```
                // Ensure that at least one WaveGenerator is kept
                if (waveGenerators.Length > 1)
                {
                    // Randomly choose an index to keep
                    int indexToKeep = Random.Range(0, waveGenerators.Length);

                    // Iterate through the WaveGenerators and destroy all except the randomly chosen one
                    for (int i = 0; i < waveGenerators.Length; i++)
                    {
                        if (i != indexToKeep)
                        {
                            Destroy(waveGenerators[i].gameObject);
                        }
                    }
                }
            }
        }
        else if (other.CompareTag("Screen") || other.CompareTag("Wall"))
        {
            // Destroy the sphere immediately upon passing through the wall
            Destroy(gameObject);
        }
    }
}
```

Codes: Observed Motion + Detector

```
using UnityEngine;

0 references
public class ScreenController : MonoBehaviour
{
    0 references
    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Particle"))
        {
            // Capture the position of the particle on the screen
            Debug.Log("Particle hit screen at position: " + other.transform.position);
        }
    }
}
```

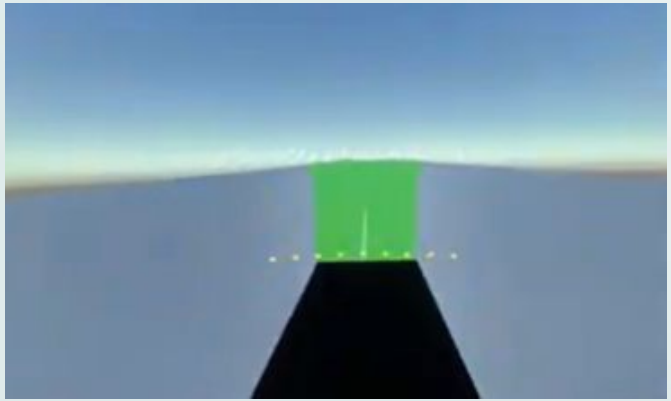
```
    void MoveObserved()
    {
        Debug.Log("this is activated");
        // Move the sphere forward along the z-axis when observed
        transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
    }

    0 references
    public void ToggleObserved()
    {
        isObserved = !isObserved;
    }
}
```

```
using UnityEngine;

0 references
public class InputMovement : MonoBehaviour
{
    1 reference
    public float moveSpeed = 5.0f;

    4 references
    private bool isObserved = false;

    0 references
    public bool IsObserved()
    {
        return isObserved;
    }

    0 references
    void Update()
    {
        if (isObserved)
        {
            MoveObserved();
        }
    }
```

```
0 references
public class CameraSwitcher : MonoBehaviour
{
    2 references
    public Camera firstCamera;
    2 references
    public Camera secondCamera;
    2 references
    public Camera thirdCamera;

    4 references
    private int currentCameraIndex = 0;

    0 references
    void Start()
    {
        // Ensure only the first camera is initially active
        SwitchCamera(currentCameraIndex);
    }

    0 references
    void Update()
    {
        // Check for user input or any condition to switch betwe
        if (Input.GetKeyDown(KeyCode.C))
        {
            // Increment the camera index and loop back to 0 if
            currentCameraIndex = (currentCameraIndex + 1) % 3;
```

```
    2 references
    void SwitchCamera(int cameraIndex)
    {
        // Disable all cameras
        firstCamera.enabled = false;
        secondCamera.enabled = false;
        thirdCamera.enabled = false;

        // Enable the selected camera
        switch (cameraIndex)
        {
            case 0:
                firstCamera.enabled = true;
                break;
            case 1:
                secondCamera.enabled = true;
                break;
            case 2:
                thirdCamera.enabled = true;
                break;
        }
    }
}
```

Day 9-10: 23rd-24th Nov



Overall Experience

Thank You