Quantum Computing Web Activity Documentation

**Index.html**
Components:
1. `<!DOCTYPE html>`: This declaration specifies the HTML version being used, which is HTML5.

2. `<html>`: This is the root element of an HTML page. It encapsulates all other elements.

3. `<head>`: This element contains metadata about the HTML page, such as the title and stylesheets.

4. `<title>`: This element sets the title of the web page, which appears in the browser's title bar or tab.

5. `<style>`: This element is used to define the styles for various elements within the web page.

6. `h1`: This is a CSS selector for the `h1` element, which represents a heading. The defined styles set the color, font size, margin, text alignment, and z-index.

7. `p`: This CSS selector targets the `p` elements, representing paragraphs. The defined styles set the text alignment and z-index.

8. `.container`: This is a CSS class selector. It targets an element with the class name `container` and defines a flex container that centers its contents horizontally.

9. `.arrow-container`: This CSS class selector targets an element with the class name `arrow-container`. It defines a flex container that centers its contents vertically and horizontally, positions it absolutely at the bottom of the page, and sets its width to 100%.

10. `.arrow`: This CSS class selector targets an element with the class name `arrow`. It defines styles for an arrow icon, including font size, color, text decoration, transition effect, and margin.

11. `.arrow:hover`: This CSS selector targets the `arrow` class when the user hovers over it. It defines a transform effect to move the arrow vertically.

12. `.arrow.back`: This CSS class selector targets an element with both `arrow` and `back` classes. It applies specific styles to the arrow when it represents a backward navigation arrow, including a transform to rotate it and adjust its margins.

13. `.arrow-text`: This CSS class selector targets an element with the class name `arrow-text`. It defines styles for a text at the bottom of the page, including color, font style, font size, margin, text alignment, and positioning.

14. `<body>`: This element represents the main content of the web page.

15. `<h1>`: This HTML element represents a heading level 1. It displays the title "Quantum Computing Experiment" in a larger font size.

16. `<p>`: This HTML element represents a paragraph. It contains text that explains the experiment and what participants can expect.

17. `<div class="arrow-container">`: This `div` element has a class name of `arrow-container`. It serves as a container for the navigation arrow.

18. `<a href="classical.html" class="arrow">&rarr;</a>`: This `a` (anchor) element represents a hyperlink. It has an `href` attribute that specifies the URL "classical.html" to navigate to when clicked. It also has a class name of `arrow`, making it stylized as an arrow icon.

19. `<p class="arrow-text">`: This `p` element has a class name of `arrow-text`. It displays text indicating that the user can click the arrows to navigate between pages.

This HTML code creates a basic web page with styles defined using CSS. It provides headings, paragraphs, and navigation elements to guide the user through a quantum computing experiment.

**classical.html**
Components:
1. `<div class="container">`: This `<div>` element has a class name of "container" and serves as a container for the bit element.

2. `<div class="bit zero" onclick="toggleBit(this)">0</div>`: This `<div>` element represents a classical bit. It has two classes: "bit" and "zero". The initial state of the bit is 0, indicated by the text content "0". The `onclick` attribute calls the JavaScript function `toggleBit(this)` when the bit is clicked.

3. `<script>`: This element is used to embed JavaScript code within the HTML document.

4. `function toggleBit(element) { ... }`: This is a JavaScript function named `toggleBit` that takes an element as an argument. It is triggered when the bit is clicked. The function checks the class list of the element to determine its current state. If the element has the class "zero", it removes that class, adds the class "one", changes the text content to "1", and visually represents the bit as the value 1. If the element has the class "one", it performs the reverse operation, changing the state from 1 to 0.

5. `<a href="index.html" class="arrow">&larr;</a>`: This `<a>` element represents a left arrow. It has an `href` attribute that specifies the URL "index.html" to navigate to when clicked. It also has a class name of "arrow" to apply styling.

6. `<a href="AND_gate.html" class="arrow">&rarr;</a>`: This `<a>` element represents a right arrow. It has an `href` attribute that specifies the URL "AND_gate.html" to navigate to when clicked. It also has a class name of "arrow" to apply styling.

7. `<p class="arrow-text">`: This `<p>` element has a class name of "arrow-text". It displays text indicating that the user can click the arrows to navigate between pages.

In summary, this code snippet creates a web page that introduces the concept of classical bits. It displays a bit with an initial state of 0, and when clicked, the bit toggles between 0 and 1 visually. The JavaScript function `toggleBit` handles the toggling logic. The page also includes navigation arrows to move to the previous and next pages, and a text description guiding the user to click the arrows for navigation.

**AND_gate.html**

Components:

1. `<div class="bit zero" id="bit1" onclick="toggleBit('bit1')">0</div>`: This `<div>` element represents the first bit. It has a class of "bit" and an id of "bit1". The initial state of the bit is 0, indicated by the text content "0". When clicked, it calls the `toggleBit` function with the argument 'bit1'.

2. `<div class="bit zero" id="bit2" onclick="toggleBit('bit2')">0</div>`: This `<div>` element represents the second bit. It has a class of "bit" and an id of "bit2". Similar to the first bit, its initial state is 0, and the `toggleBit` function is called with the argument 'bit2' when clicked.

3. `<div class="gate and" onclick="applyGate()">AND</div>`: This `<div>` element represents the logical AND gate. It has a class of "gate" and "and". When clicked, it triggers the `applyGate` function.

4. `<div class="bit zero" id="result">0</div>`: This `<div>` element represents the result of applying the AND gate. It has a class of "bit" and an id of "result". Initially, the result is 0.

5. `function toggleBit(id) { ... }`: This JavaScript function handles the toggling of a bit's state. It takes an id as an argument to identify which bit is being toggled. It finds the element with the specified id, checks its current state, updates the state and visual representation, and modifies the innerHTML accordingly.

6. `function applyGate() { ... }`: This JavaScript function applies the logical AND operation to the two input bits and updates the result. It retrieves the bit elements and the result element by their respective ids. If both input bits are in the state of 1, it updates the result to 1. Otherwise, it sets the result to 0.

The rest of the code remains similar to the previous example, including the styling and navigation elements.

In summary, this web page demonstrates a logical AND gate using HTML, CSS, and JavaScript. Users can toggle the input bits by clicking on them and compute the result by clicking the AND gate button. The result is dynamically updated based on the input bit states.

**OR_gate.html**
Components:
1. `<div class="bit zero" id="bit1" onclick="toggleBit('bit1')">0</div>`: This `<div>` element represents the first bit. It has a class of "bit" and an id of "bit1". The initial state of the bit is 0, indicated by the text content "0". When clicked, it calls the `toggleBit` function with the argument 'bit1'.

2. `<div class="bit zero" id="bit2" onclick="toggleBit('bit2')">0</div>`: This `<div>` element represents the second bit. It has a class of "bit" and an id of "bit2". Similar to the first bit, its initial state is 0, and the `toggleBit` function is called with the argument 'bit2' when clicked.

3. `<div class="gate or and" onclick="applyGate()">OR</div>`: This `<div>` element represents the logical OR gate. It has a class of "gate", "or", and "and". When clicked, it triggers the `applyGate` function.

4. `<div class="bit zero" id="result">0</div>`: This `<div>` element represents the result of applying the OR gate. It has a class of "bit" and an id of "result". Initially, the result is 0.

5. `function toggleBit(id) { ... }`: This JavaScript function handles the toggling of a bit's state. It takes an id as an argument to identify which bit is being toggled. It finds the element with the specified id, checks its current state, updates the state and visual representation, and modifies the innerHTML accordingly.

6. `function applyGate() { ... }`: This JavaScript function applies the logical OR operation to the two input bits and updates the result. It retrieves the bit elements and the result element by their respective ids. If either input bit is in the state of 1, it updates the result to 1. Otherwise, it sets the result to 0.

The rest of the code remains similar to the previous examples, including the styling and navigation elements.

In summary, this web page demonstrates a logical OR gate using HTML, CSS, and JavaScript. Users can toggle the input bits by clicking on them, and the result is dynamically updated based on the OR operation.

**hadamard.html**
This code snippet demonstrates a simple representation of a qubit and a Hadamard gate using HTML, CSS, and JavaScript. Let's break down the components:

1. `<div class="qubit" id="qubit" draggable="true" ondragstart="drag(event)" onclick="toggleBit(this)">0</div>`: This `<div>` element represents the qubit. It has a class of "qubit" and an id of "qubit". The initial state of the qubit is 0, indicated by the text content "0". It is draggable and triggers the `drag` function on drag events and the `toggleBit` function on click events.

2. `<div class="left-box box" id="left-box" ondrop="drop(event)" ondragover="allowDrop(event)"></div>`: This `<div>` element represents the left box where the qubit can be dropped. It has a class of "box" and an id of "left-box". It triggers the `drop` function on drop events and the `allowDrop` function on dragover events.

3. `<div class="right-box box" id="right-box"></div>`: This `<div>` element represents the right box where the qubit is placed after being dragged. It has a class of "box" and an id of "right-box".

4. `<div class="gate" id="hadamard">H</div>`: This `<div>` element represents the Hadamard gate. It has a class of "gate" and an id of "hadamard". The gate is represented by the letter "H".

5. `<div class="horizontal-line" id="horizontal-line" style="left: calc(35% + 62px); width: calc(50% - 35% - 76px);"></div>`: This `<div>` element represents the horizontal line connecting the qubit and the Hadamard gate. It is positioned using CSS styling.

6. `<script> ... </script>`: This script section contains JavaScript functions for handling the qubit manipulation and Hadamard gate operations. The functions include `toggleBit`, `superpositionState`, `drag`, `allowDrop`, `drop`, `applyHadamard`, `applyInverseHadamard`, `resetQubit`, and `measureQubit`. These functions handle the toggling of qubit states, drag and drop events, applying the Hadamard gate, resetting the qubit, and measuring the qubit state.

7. `<button onclick="resetQubit()" style="position: absolute; top: 50%; right: 5%; z-index: 5;">Reset</button>`: This button triggers the `resetQubit` function when clicked and is positioned using CSS styling.

8. `<button onclick="measureQubit()" style="position: absolute; top: 50%; right: 10%; z-index: 5;">Measure</button>`: This button triggers the `measureQubit` function when clicked and is positioned using CSS styling.

9. `<div class="arrow-container"> ... </div>`: This `<div>` element contains the navigation arrows for previous and next pages.

10. `<p class="arrow-text"> Click arrows to navigate pages</p>`: This `<p>` element displays a text indicating to click the arrows for page navigation.

In summary, this web page represents a qubit and a Hadamard gate, allowing users to toggle the qubit state, apply the Hadamard gate by dragging and dropping the qubit onto the gate, reset the qubit state, and measure the qubit state. The page provides buttons for resetting and measuring the qubit, along with navigation.

**CNOT.html**

Components:

1. `<div class="qubit control zero" id="control-qubit" data-top="40%" draggable="true" ondragstart="drag(event)" onclick="toggleBit(this)">0</div>`: This `<div>` element represents the control qubit. It has a class of "qubit", "control", and "zero". The initial state of the qubit is 0, indicated by the text content "0". It is draggable and triggers the `drag` function on drag events and the `toggleBit` function on click events.

2. `<div class="qubit target zero" id="target-qubit" data-top="60%" draggable="true" ondragstart="drag(event)" onclick="toggleBit(this)">0</div>`: This `<div>` element represents the target qubit. It has a class of "qubit", "target", and "zero". Similar to the control qubit, its initial state is 0, and it triggers the `drag` function on drag events and the `toggleBit` function on click events.

3. `<div class="gate" id="cnot" style="position: absolute; top: 50%; left: 50%; width: 70px; transform: translate(-50%, -50%);">CNOT</div>`: This `<div>` element represents the CNOT gate. It has a class of "gate" and an id of "cnot". It is positioned at the center of the page using CSS styling.

4. `<div class="box control-left-box" id="control-left-box" ondrop="drop(event, 'control')" ondragover="allowDrop(event)"></div>`: This `<div>` element represents the left box where the control qubit can be dropped. It has a class of "box" and an id of "control-left-box". It triggers the `drop` function on drop events and the `allowDrop` function on dragover events.

5. `<div class="box control-right-box" id="control-right-box"></div>`: This `<div>` element represents the right box where the control qubit is placed after being dragged. It has a class of "box" and an id of "control-right-box".

6. `<div class="box target-left-box" id="target-left-box" ondrop="drop(event, 'target')" ondragover="allowDrop(event)"></div>`: This `<div>` element represents the left box where the target qubit can be dropped. It has a class of "box" and an id of "target-left-box". It triggers the `drop` function on drop events and the `allowDrop` function on dragover events.

7. `<div class="box target-right-box" id="target-right-box"></div>`: This `<div>` element represents the right box where the target qubit is placed after being dragged. It has a class of "box" and an id of "target-right-box".

8. `<div class="vertical-line" id="vertical-line" style="left: calc(50% - 45px); top: 50%; height: calc(20% + 5px);"></div>`: This `<div>` element represents the vertical line connecting the control and target qubits. It is positioned using CSS styling.

9. `<div class="horizontal-line" id="horizontal-line" style="left: calc(45% + 10px); top: 40%; width: calc(10% - 110px);"></div>`: This

`<div>` element represents the horizontal line connecting the control qubit and the CNOT gate. It is positioned using CSS styling.

10. `<script> ... </script>`: This script section contains JavaScript functions for handling the qubit manipulation and CNOT gate operations. The functions include `toggleBit`, `superpositionState`, `drag`, `allowDrop`, `drop`, `moveQubit`, `applyHadamard`, `applyInverseHadamard`, `applyCNOT`, `resetQubits`, `resetQubit`, and `measureQubit`. These functions handle the toggling of qubit states, drag and drop events, applying the Hadamard gate, applying the inverse Hadamard gate, applying the CNOT gate, resetting the qubits, and measuring the qubit state.

11. `<button onclick="resetQubits()" style="position: absolute; top: 50%; right: 5%; z-index: 5;">Reset</button>`: This button triggers the `resetQubits` function when clicked and is positioned using CSS styling.

12. `<div class="arrow-container"> ... </div>`: This `<div>` element contains the navigation arrows for previous and next pages.

13. `<p class="arrow-text"> Click arrows to navigate pages</p>`: This `<p>` element displays a text indicating to click the arrows for page navigation.

In summary, this web page represents two qubits and a CNOT gate, allowing users to drag and drop qubits, toggle their states, and observe the effects of applying the CNOT gate. The page also provides a reset button for restoring the initial state of the qubits.

**hadamardCNOT.html**

Components:

1. `<h1>Qubits, Hadamard Gate and CNOT Gate</h1>`: This heading element displays the title.

2. `<p>`: This paragraph element provides a description of how to entangle two qubits using the Hadamard gate and CNOT gate. It also mentions the concept of Bell states and their importance in quantum computing.

3. `<div class="qubit control zero" id="control-qubit" ...>` and `<div class="qubit target zero" id="target-qubit" ...>`: These `<div>` elements represent the control and target qubits, respectively. They have classes of "qubit", "control" and "target" to distinguish them. Initially, they both display the state 0. Clicking on a qubit triggers the `toggleBit` function to toggle its state between 0 and 1.

4. `<div class="gate" id="cnot" ...>CNOT</div>`: This `<div>` element represents the CNOT gate.

5. `<div class="box control-left-box" ...>` and `<div class="box control-right-box" ...>`: These `<div>` elements serve as drop zones for the control qubit in the left and right boxes.

6. `<div class="box target-left-box" ...>` and `<div class="box target-right-box" ...>`: These `<div>` elements serve as drop zones for the target qubit in the left and right boxes.

7. `<div class="vertical-line" ...>`: These `<div>` elements represent vertical lines connecting the control and target qubits to the CNOT gate.

8. `<div class="horizontal-line" ...>`: These `<div>` elements represent horizontal lines connecting the boxes and gates.

9. `<script> ... </script>`: This script section contains various JavaScript functions to handle the drag and drop functionality, applying Hadamard gate, applying CNOT gate, resetting qubits, and measuring qubits.

10. `<button onclick="resetQubits()" ...>` and `<button onclick="measureQubit()" ...>`: These buttons trigger the `resetQubits` and `measureQubit` functions, respectively, when clicked.

11. `<div class="arrow-container"> ... </div>`: This `<div>` element contains the navigation arrow for the previous page.

12. `<p class="arrow-text"> Click arrows to navigate pages</p>`: This `<p>` element displays a text indicating to click the arrow for page navigation.

In summary, this web page demonstrates the use of the Hadamard gate and CNOT gate to entangle two qubits. The user can interact with the qubits by clicking on them to toggle their states between 0 and 1. The qubits can be dragged and dropped into the appropriate boxes to apply the gates. There are also buttons to reset the qubits and measure their states.