**Plan to Fix the Bugs**

The first error we found in the game is that the user can drag and drop more than one piece into the designated drop location. The pieces end up overlapping, which shouldn't happen—there should only be a single piece in the drop location at a time.

The code for this part initially looks like this:

function handleDrop(e) {

e.preventDefault();

console.log("dropped something on me");

this.appendChild(draggedPiece);

}

By analyzing and studying this part of the code, I understood that this, when used in the handleDrop function, represents the location where the piece was dropped, meaning the drop zone. The appendChild method is used to move one element into another. Therefore, the code is moving the draggedPiece element into this, which is the drop zone. The problem is that appendChild is moving the piece into the drop zone without checking whether there is already another element there.

**Solution**: we need to add a check to see if there is already an element in the drop zone before allowing a new piece to be added. The .hasChildNodes() method is used precisely for this—it checks whether an element has child nodes. I chose .hasChildNodes() because it is a simpler and more direct way to verify whether there is already a piece inside the drop zone. If the function returns true, it means there is already a piece there, preventing another one from being added. Other alternatives could be childElementCount, which returns the number of child elements inside the drop zone. However, .hasChildNodes() is more intuitive because it directly returns a boolean (true or false), whereas childElementCount requires a numerical comparison. firstChild or firstElementChild, which could also be checked to determine if an element exists inside the drop zone. This would also work, but .hasChildNodes() checks for any type of child node, not just elements, making the code more generic and safer. We can correct the error as follows: READ HERE

function handleDrop(e) {

e.preventDefault();

console.log("dropped something on me");

if (!this.hasChildNodes()) {

this.appendChild(draggedPiece);

}

}

The second error found in the game occurs when the player tries to reset the game to assemble a new puzzle, but the pieces do not return to their original location.

The changeBGImage function is responsible for changing the puzzle background whenever it is called. The puzzleBoard.style.backgroundImage code accesses the puzzleBoard element and changes its style directly via JavaScript, modifying the backgroundImage CSS property. This part of the code url(images/backGround${this.id}.jpg) links JavaScript with CSS, where ${this.id} references the element that activated the function. This way, url(images/backGround${this.id}.jpg) dynamically creates the background image URL by replacing ${this.id} with the element's ID.

The error in this code is that it only changes the background image but does not interact with the pieces that have already been placed in the drop zones when selecting a new puzzle.

**Solution**: To return the pieces to their original location, we can use the following code:

let puzzleContainer = document.querySelector(".puzzle-pieces");

Here, puzzleContainer selects the HTML element with the class .puzzle-pieces, which is the area where the pieces initially reside. The querySelector method selects the first element that matches the provided CSS selector. In this case, it selects the element with the .puzzle-pieces class, which is where the puzzle pieces should be returned after being removed from the drop zones.

The second part of the code to solve the problem is this:

dropZones.forEach((zone) => {

if (zone.firstChild) {

puzzleContainer.appendChild(zone.firstChild);

}

});

Here, the forEach function is used to loop through all the zones where the player can drop the pieces (zone). The code if (zone.firstChild) checks if there is any piece in the drop zone. The firstChild property returns the first child of the drop zone. If a piece is already in the drop zone, the check will be true, and the code inside the if statement will execute.

With the code puzzleContainer.appendChild(zone.firstChild) if the drop zone has a piece, the first piece will be moved back to the starting area where the pieces are available for the player to drag again. The appendChild method repositions the piece at the last position within puzzleContainer.