**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 to the designated drop location. The pieces end up on top of each other and this shouldn't happen, there should only be a single piece in the place where we release the piece at a time.

The code for this part, in particular, initially looks like this:

function handleDrop(e) {

  e.preventDefault();

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

  this.appendChild(draggedPiece);

}

Studying this part of the code, I understood that this, when used in the handleDrop function, represents the location where the part was dropped, that is, the dropzone. appendChild is used to move one element into another. So the code is saying to move the draggedPiece element into this which is the drop-zone. Apparently what is happening is that appendChild is moving the element to the drop-zone without checking whether or not there is another element there.

**Solution**: add code that checks whether or not an element already exists in the drop-zone. The .hasChildNodes code serves precisely to say that, if you have one! It means there's already a piece there. In this case we can do it as follows:

function handleDrop(e) {

  e.preventDefault();

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

  if(!this.hasChildNodes())

  this.appendChild(draggedPiece);

}

The second error found in the game is when the player tries to reset the game to do another puzzle and the pieces do not return to their original place.

The function changeBGImage code is responsible for changing the background of the puzzle whenever called. The puzzleBoard.style.backgroundImage code accesses the puzzleBoard element and changes its style. The style will allow CSS properties to be changed directly via JS. What defines the background image is the css backgroundImage property. This part of the code `url(images/backGround${this.id}.jpg)` links the js with the css, the ${this.id} references the element that activated the function. Then url(images/backGround${this.id}.jpg) creates the background image URL, replacing ${this.id} with the element ID, generating a dynamic image URL. The error in this code is that it only changes the background image, but does not interact with the pieces in the drop-zone when selecting a new puzzle.

**Solution**: to return the parts to their original location, we can use the following codes:

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

Here the puzzleContainer will select the HTML element with the class .puzzle-pieces which is the area where the pieces initially reside. With the querySelector we will select the first element corresponding to the provided CSS. In this case, it selects the element with the puzzle-pieces class, which is where puzzle pieces can be replenished after being removed from the drop-zones.

The second code to solve the problem is this:

dropZones.forEach((zone) => {

if (zone.firstChild) {

puzzleContainer.appendChild(zone.firstChild); }

Here the forEach function is used to traverse the zones that the player drops the pieces (zone). With the code if (zone.firtsChild) we can check if there is any part in the drop-zone. The firstChild property returns the first child of the drop-zone. If a part already exists in the drop-zone, the check will be true and the code inside the if will be executed. With the code puzzleContainer.appendChild(zone.firstChild), if the drop-zone has any pieces, it will move the first piece from the drop-zone back to the area where the pieces are for the player to drag. The appendChild function moves the piece to the last position within the puzzleContainer.