

## Assignment 3 – JavaScript Game

### Description:

For my project I decided to do a puzzle game. In this game you are supposed to drag and drop tiles inside the blank tile to move the tiles around to solve the image. You are only able to change tiles that are adjacent to the blank tile. You CANNOT drag and drop tiles that are diagonally placed.

### Approach / What I Did:

The first thing I did was try and figure out what type of game I wanted to make. When I decided to make the puzzle game I had an idea of how to start, but I still needed some help. What I did I looked at multiple videos of people doing a similar game and took note how different people took different approaches. The way that I approached doing this assignment was thinking about it like writing descriptive directions. I started by making it 9 tiles total

```
const numRows = 3;  
const numColumns = 3;
```

I also had to make sure that tiles only swapped if they were next to each other, and not swap if they are diagonally placed.

```
function handleDragEnd() { //checks where the dragged tile is and if it is next to it it will  
  if (!blankTile.src.includes("9.jpg")) {  
    /**this is blank tile it will stop function if not  
    * dragged in the correct spot */  
    return;  
  }  
  //checks where tiles are in the puzzle  
  const [currentRow, currentCol] = currentTile.id.split("-").map(Number);  
  const [blankRow, blankCol] = blankTile.id.split("-").map(Number);  
  
  const isAdjacent = ( //checks if the blank tile is next to the current tile  
    (currentRow === blankRow && Math.abs(currentCol - blankCol) === 1) ||  
    (currentCol === blankCol && Math.abs(currentRow - blankRow) === 1)  
  );  
  if (isAdjacent) { //if it is adjacent then you can swap tiles  
    swapTiles(currentTile, blankTile);  
  }  
}
```

**Issues and Resolutions:**

One of the issues that I first came across was making an unsolvable game. Depending on the array and where the images in the array were, the puzzle would actually be unsolvable.

```
const shuffledImageOrder = ["4", "2", "8", "5", "1", "6", "3", "7", "9"];
```

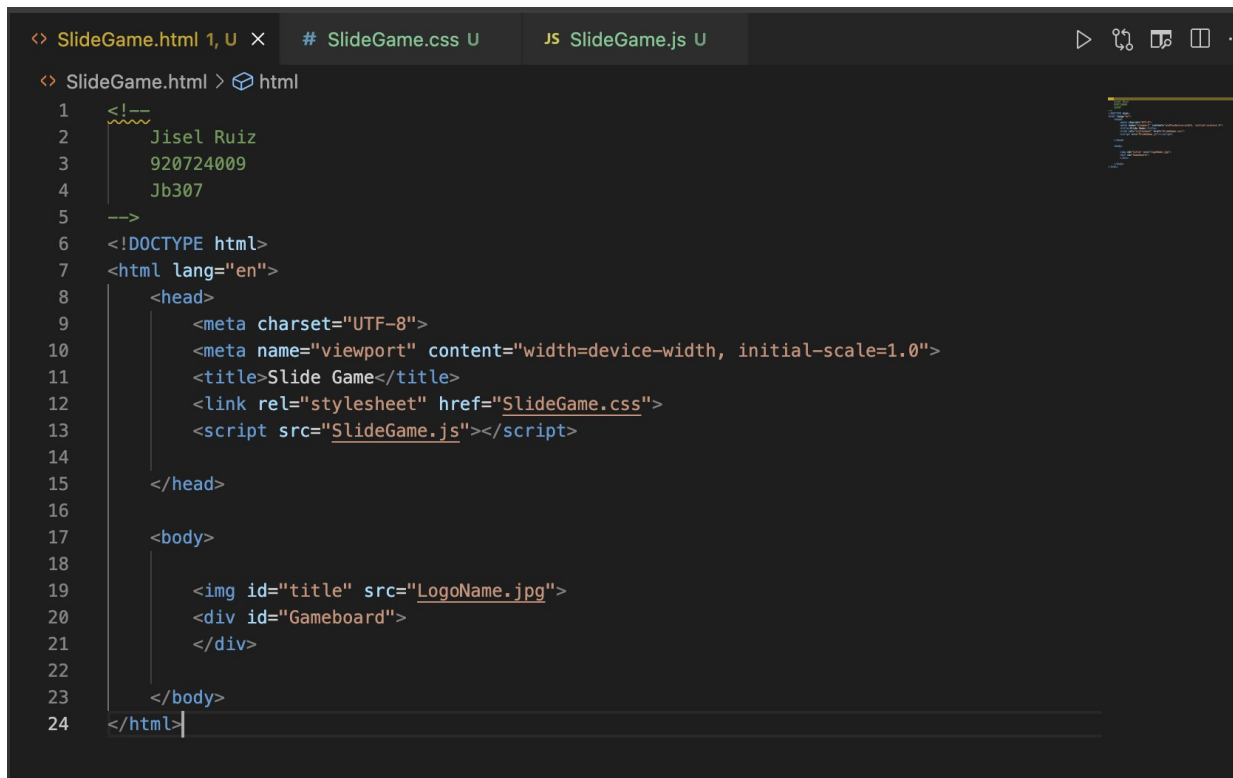
What I did to solve this was by trial and error. I would play the game and if I could not get it then I would change the array. The only thing that stayed constant was the 9 at the end because that is the blank tile.

**Analysis:** (Reflect on what you have learned and how decisions made could impact users.)

This project was fun and it made me think of coding as a set of precise directions that can impact the outcome of the assignment. I enjoyed how visual it was for me. I was able to make this assignment a bit simpler, (because it was a bit difficult) by thinking about all the steps needed in order to create a puzzle like this, for example where the tiles go, how they drag/drop, and if they are allowed to be placed in a certain tile.

One of the things that could potentially impact users is the drag and drop feature, it could be hard to drag and drop the puzzle for a long time, especially if it is a harder puzzle. One way that I thought about fixing this issue is instead of using the mouse to drag and drop, I could make it compatible with the keyboard and the user has the choice of using the mouse or the arrow keys for a more seamless experience.

**Screen shots:**



```
<> SlideGame.html 1, U x # SlideGame.css U JS SlideGame.js U
<> SlideGame.html > html
1  <!--
2      Jisel Ruiz
3      920724009
4      Jb307
5  -->
6  <!DOCTYPE html>
7  <html lang="en">
8      <head>
9          <meta charset="UTF-8">
10         <meta name="viewport" content="width=device-width, initial-scale=1.0">
11         <title>Slide Game</title>
12         <link rel="stylesheet" href="SlideGame.css">
13         <script src="SlideGame.js"></script>
14     </head>
15     <body>
16
17         
18         <div id="Gameboard">
19             </div>
20     </body>
21 </html>
```

```
# SlideGame.css > #Gameboard img
1  /**
2   Jisel Ruiz
3   920724009
4   Jb307
5   */
6
7  body {
8      font-family: Arial, Helvetica, sans-serif;
9      text-align: center;
10     color: #771da4;
11     min-height: 100vh;
12     background-image: linear-gradient(■ rgb(236, 247, 141), ■ rgb(92, 34, 127));
13 }
14
15
16
17 #title {
18     height: 350px;
19     width: 400px;
20 }
21
22
23 #Gameboard {
24     width: 360px;
25     height: 360px;
26     background-color: ■ rgb(168, 64, 184);
27     border: 5px solid ■ #771da4;
28
29     margin: 0px auto; /*centers */
30     display: flex;
31     flex-wrap: wrap;
32 }
33
34 #Gameboard img {
35     /*the sizes of the squares*/
36     width: 118px;
37     height: 118px;
38     border: 1px solid ■ #1f0543; /*lines between pictures*/
39 }
```

```
47
48 function handleDragStart() { //sets the current tile to tile being dragged
49     currentTile = this;
50 }
51
52 function handleDragOver(event) {
53     event.preventDefault();
54 }
55
56 function handleDragEnter(event) {
57     event.preventDefault();
58 }
59
60 function handleDragLeave() { //placeholder
61 }
62
63
64 function handleDragDrop() { //sets blank tile to tile that was dropped
65     blankTile = this;
66 }
67
68 function handleDragEnd() { //checks where the dragged tile is and if it is next to it it will
69     if (!blankTile.src.includes("9.jpg")) {
70         /**this is blank tile it will stop function if not
71         * dragged in the correct spot **/
72         return;
73     }
74     //checks where tiles are in the puzzle
75     const [currentRow, currentCol] = currentTile.id.split("-").map(Number);
76     const [blankRow, blankCol] = blankTile.id.split("-").map(Number);
77
78     const isAdjacent = ( //checks if the blank tile is next to the current tile
79         (currentRow === blankRow && Math.abs(currentCol - blankCol) === 1) ||
80         (currentCol === blankCol && Math.abs(currentRow - blankRow) === 1)
81     );
82     if (isAdjacent) { //if it is adjacent then you can swap tiles
83         swapTiles(currentTile, blankTile);
84     }
85 }
86 }
87
```

```
JS SlideGame.js > ...
1  //Jisel Ruiz
2  //920724009
3  //Jb307
4
5  //this defines the rows and col of the board, 9 squares total
6  const numRows = 3;
7  const numColumns = 3;
8
9  //this an array that holds the images in the order that they are going to be displayed
10 const shuffledImageOrder = ["4", "2", "8", "5", "1", "6", "3", "7", "9"];
11
12 window.onload = initializeGameboard;
13 //this function goes through each tile in the puzzle
14 function initializeGameboard() {
15     for (let row = 0; row < numRows; row++) { //moves though each row in puzzle
16         for (let col = 0; col < numColumns; col++) { //moves through each column
17             createAndAppendTile(row, col);
18         }
19     }
20 }
21
22 function createAndAppendTile(row, col) {
23     const tile = createTileElement(row, col);
24
25     tile.addEventListener("dragstart", handleDragStart); //when you drag
26     tile.addEventListener("dragover", handleDragOver); //over the tile
27     tile.addEventListener("dragenter", handleDragEnter); //when you enter tile
28     tile.addEventListener("dragleave", handleDragLeave); //when you leave tile
29     tile.addEventListener("drop", handleDragDrop); //when you drop something in the tile
30     tile.addEventListener("dragend", handleDragEnd); //when drag ends
31
32     appendTileToGameboard(tile);
33 }
34
35 function createTileElement(row, col) {
36     const tile = document.createElement("img");
37     tile.id = `${row}-${col}`; //img identifier
38     tile.src = `${shuffledImageOrder.shift()}.jpg`; //sets img for tile
39
40     return tile;
41 }
42
43 function appendTileToGameboard(tile) {
44     document.getElementById("Gameboard").append(tile);
45 }
```

```
87
88 //this function will swap the images of the two tiles
89 function swapTiles(tile1, tile2) {
90     const tempSrc = tile1.src;
91     tile1.src = tile2.src;
92     tile2.src = tempSrc;
93 }
94
```

**SAN FRANCISCO**  
**MATCH**



