CSC 4370 Project 3 Presentation

ULYSSES CARLOS

Project Roles

- Ulysses Carlos
  - Main Programmer and Designer

The Problem

- Create a Sliding Puzzle using JavaScript (Including DOM), HTML and CSS
  - Use at least one image
  - Clicking on a tile bordering the empty tile will move the tile to that position
  - Clicking on the blank tile/surrounded tile will produce no effect

Solution

- Used HTML, CSS, and Javascript to finish the project
  - HTML
    - The Webpage itself
  - CSS
    - Controlled Page Background and table
  - JavaScript
    - Displayed Puzzle pieces
    - Implementated Puzzle Randomization, Solve For Me!, and Difficulty Level

The Website

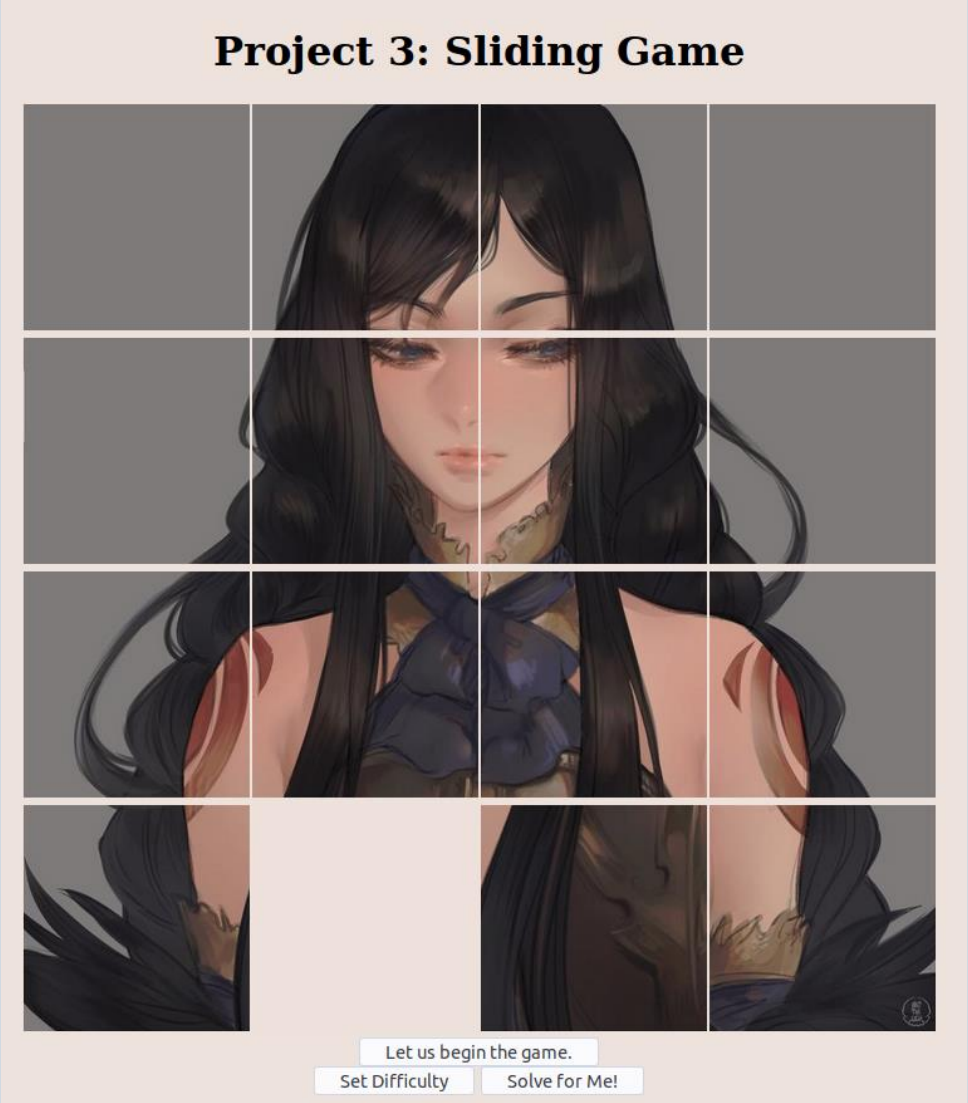
- "Let us begin the Game" executes main() in main.js
- Difficulty level can be changed by pressing the "Set Difficulty" button
- The "Solve for Me!" Button will solve the puzzle regardless if you have started the puzzle or not.

The Website (cont)

- An inline script plays a song throught the puzzle. It does not loop and is randomly choosen on each refresh.
- The image displayed for each puzzle is chosen at random out of a collection of 10 different images.

The Website(cont)

- Each image has been broken down into 16 250x 250 tiles, which can be downscaled to any resolution.



Main()

- Pressing "Let us start the game" will call the function `initalize_current_image()` which sets the image tiles, randomizes it, initializes the puzzle history, and stores the solved image in an array.

```
////////////////////////////////////////////////////////////////////////////////
// Main
////////////////////////////////////////////////////////////////////////////////

function initialize_current_image(){
    current_image_list = [];

    image_index = rand_int(0, image_folder_list.length);
    current_image = image_folder_list[image_index];
    // Now populate
    draw_table();
    td_list = document.getElementsByTagName("img");
    // Set the last one to be the blank:
    //blank_index = td_list.length - 1;
    blank_index = rand_int(0, max_images);
    set_blank(blank_index);

    correct_image_list = current_image_list.slice();

    // Initialize History
    puzzle_history = [];
    puzzle_history.push(blank_index);


    setTimeout(shuffle_image, 1000 * 3);
    // Print History


    setTimeout(print_history, 1000 * 4);
    //player_history = [];
    /*
    setTimeout(function(){
        player_history.push(puzzle_history[puzzle_history.length -1]);
    }, 2000);
    */

}
```

Draw_table()

- An image is selected at random, and each segment is pushed into current_image_list
- Each image has an onclick event `move()` which passes a value alongside it.
- The image is stored in a table datum, which is appended into a row.
- This row is then appended to the table.

```
364
365 ////////////////////////////////////////////////////////////////////////////////
366 // Draw Functions
367 ////////////////////////////////////////////////////////////////////////////////
368 function draw_table(){
369     table = document.getElementById("image_table");
370     table.innerHTML = "";
371     var row;
372     var table_d;
373     var temp_image;
374     var image_name;
375     var num;
376
377     for (var i = 0; i < row_len; i++){
378         row = document.createElement("tr");
379         for (var j = 0; j < column_len; j++){
380             temp_image = document.createElement("img");
381             num = column_len * i + j;
382             image_name = current_image + "/image-" + j + "-" + i + ".jpg";
383             temp_image.setAttribute("src", image_name);
384             current_image_list.push(image_name);
385             table_d = document.createElement("td");
386             table_d.appendChild(temp_image);
387             row.appendChild(table_d);
388             // temp_image.onclick = handle(num);
389             temp_image.onclick = move_wrapper(num);
390         }
391         table.insertBefore(row, table.childNodes[i + 1]);
392     }
393
394 }
395
```

Shuffle()

- From the blank tile, shuffle() shuffles the tiles based on difficulty.
- The program randomly moves a direction and then determines if the move is valid.
- If the move is valid, it adds a value to puzzle_history and swaps the two tiles.

```
function move_blank(cardinal){
    var check;
    var temp;
    switch (cardinal){
        case 0: // Up
            temp = blank_index + row_len;
            check = move_up_check(temp);
            break;
        case 1: // Down
            temp = blank_index - row_len;
            check = move_down_check(temp);
            break;
        case 2: // Left:
            temp = blank_index  + 1;
            check = move_left_check(temp);
            break;
        case 3: // Right
            temp = blank_index - 1;
            check = move_right_check(temp);
            break;
        default:
            check = false; // Just do that.
    }

    if (check && is_in_bounds(temp)){
        puzzle_history.push(temp);
        move_to_blank(temp, blank_index);
    }

}

function shuffle_image(){

    var moves = difficulty;
    var rand;
    var directions = 4;
    var moved_blank = blank_index;
    for (var i = 0; i < moves; i++){
        rand = rand_int(0, directions);
        move_blank(rand);
        moved_blank = blank_index;
    }
    //player_history.push(puzzle_history[puzzle_history.length - 1 ]);
}
```

Move()

- Accepts the parameter index, which is generated for each tile.
- It determines if the tile can move by checking if any of its neighbors is a blank tile through the function check_neighbors
- If a neighbor is a blank tile, it swaps both tiles, giving the impression that the tile has moved.

```
154
155  function check_neighbors(index){
156      // There can only be one value;
157
158      if (move_up_check(index))
159          return index - row_len;
160      else if (move_down_check(index))
161          return index + row_len;
162      else if (move_left_check(index))
163          return index - 1;
164      else if (move_right_check(index))
165          return index + 1;
166      else return -1; // Not possible.
167
168
169  }
170
171  function move(index){
172  //    console.log("Checking if image[" + index + "] can move  ... \n");
173      var value = check_neighbors(index);
174      if (value =/= -1){
175          //player_history.push(index);
176          puzzle_history.push(index);
177          move_to_blank(index, value);
178          check_winning_condition();
179      }
180      else
181          console.log("Can't move image[" + index + "] anywhere ... ");
182
183  }
184
185  function check_winning_condition(){
186      var check;
187      for (var i = 0; i < correct_image_list.length; i++)
188          if (current_image_list[i] =/= correct_image_list[i]) return false;
189
190      // if true // , now display an alert button to clear screen:
191
192      window.alert("Congratulations, you have won the game.");
193      location.reload();
194
195  }
```

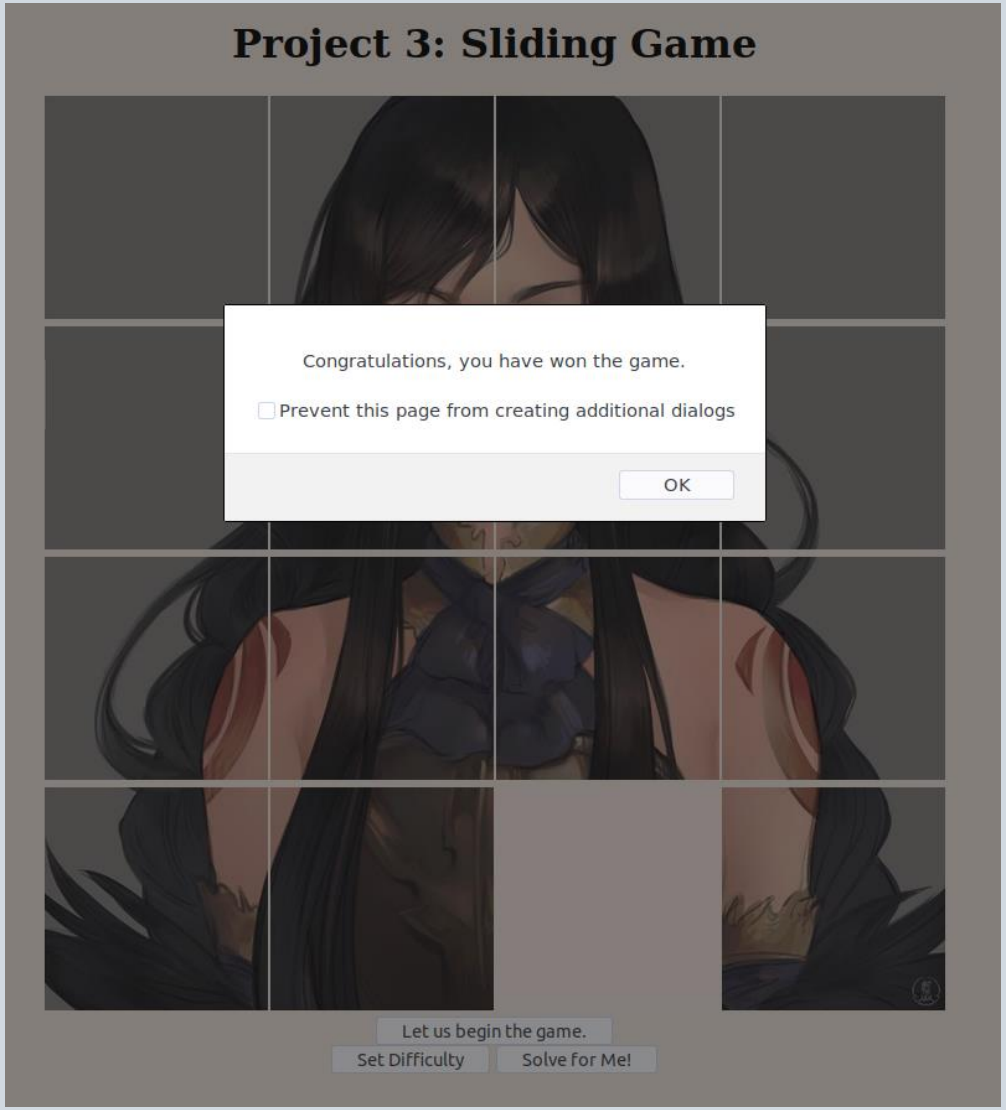
Move() (cont.)

- `check_winning_condition()` is executed at the end of move() to determine if the puzzle has been solved.
- The array `correct_image_list` is then compared to the current_image_list.
- The game is won if both arrays are equal.

```
185  function check_winning_condition(){
186      var check;
187      for (var i = 0; i < correct_image_list.length; i++)
188          if (current_image_list[i] != correct_image_list[i]) return false;
189
190      // if true // , now display an alert button to clear screen:
191
192      window.alert("Congratulations, you have won the game.");
193      location.reload();
194
195  }
196
```

Move() (cont.)

- A popup is displayed when the game is solved
- Closing the popup will reload the page.



Difficulty

- Clicking on "Set Difficulty" will open a popup that can change the difficulty level
- The difficulty level will change the amount of moves that the program can use to randomize the image.

Solve_for_me()

- Clicking on "Solve For Me" will solve the puzzle regardless if the player has moved tiles or not.
- It activates solve_for_me(), which uses the array puzzle_history to backtrace back into the default image.

```
async function solve(){

    // First check if player history is empty//
    /*
    if (player_history.length ≠ 0){
        for (var i = player_history.length - 1; i ⩾ 1; i--)
        swap(player_history[i], player_history[i - 1]);
        await sleep(300);
    }
*/
    for (var i = puzzle_history.length - 1; i ⩾ 1; i--){
        swap(puzzle_history[i], puzzle_history[i - 1]);
        await sleep(300);
    }

    await sleep (1000);
    window.alert("Problem solved. Reloading in 3 seconds.");
    await sleep (3000);
    location.reload();
}

function solve_for_me(){
    // Using the history list:

    if (puzzle_history ⩽ 0){
        window.alert("I haven't randomized the puzzle yet. Chill.")
        return;
    }

    solve();
}
```