**CHOOSE DOOR**

Your mission is to construct a single-page website that plays a fully-functional game. You will see how HTML, CSS, and JavaScript interact harmoniously to produce a dynamic website.

Believe it or not, you have the capabilities of building every feature in this game - from the layout to the logic. Be patient, take creative risks, and most importantly, have fun!

**1.**

You’ll build out functionality one step at a time in **index.html**, **style.css**, and **script.js**. Take a moment to familiarize yourself with what’s already in your project to start.

**2.**

Now your first objective is to build one door which will hide the dreaded FindFly.

Look at the **index.html** starting code. Inside the <body>, create a <div> parent element with the class name "door-row". Next, create <img> child element inside this <div>.

Hint: Make sure your HTML elements are properly nested:

**3.**

In Project folder; use door\_closed image inside <img> element’s src.

Refresh the page to display the closed doors for your game.

**4.**

A user should know that this closed door can be clicked.

Assign this <img> element an id attribute with the value of door\_a. Place these id before the src.

Hint: Make sure your id value is wrapped in quotation marks:

**5.**

Connecting this id to the CSS will allow you to adjust the closed door’s properities - including changing the cursor when you hover over it.

Navigate to the **style.css** starting code and create an CSS ID selector for the #door\_a. Inside this selector underneath the body selector, assign the cursor property a value of pointer.

Refresh the page and notice that the cursor changes to a pointer as soon as you hover over the door.

**6.**

The pointer’s purpose is to indicate to the user that the closed door image can be clicked. **CSS** made this possible; but it’s the **JavaScript** that makes that click do something.

All **JavaScript** logic will be written within the script.js file but in order to have that logic interact with the **HTML**, you first need to establish a connection between the **script.js** page and the **index.html**. In the **index.html** page, create a <script> element directly above the closing </body> tag. Inside the opening <script> tag, set the type as "text/javascript" and the src as "script.js". Then immediately close the element with a </script> tag.

**7.**

Navigate to the **script.js** file. Create a global variable called doorImage1. Use a JavaScript DOM method to assign this global variable to the HTML element with the id of door\_a.

Hint: Use the .getElementById() DOM method.

**8.**

Inside the **script.js** file, underneath your global variable, assign doorImage1.onclick to a new, empty arrow function.

This function will run whenever the first door image element is clicked.

**9.**

Now make the closed door image change when you click it so that you see an open door with the FindFly.

First, directly underneath your doorImage1 global variable, create a new global variable called flyDoorPath. Set its value to the “fly.png” image path inside your folder

Hint: Make sure to wrap the URL path in quotes:

**10.**

Next, within your doorImage1.onclick arrow function, change the src of doorImage1 to the value of flyDoorPath.

Refresh the page. Now when you click on the door, watch as the closed door image changes to the FindFly ready to greet you with housework!

Hint

A variable’s src value can be assigned a new value held by another variable:

variableA.src = path;

**11.**

You’ve completed your first objective of getting a door and opening a door. Now you’re going to add two more closed doors in the same parent <div> as your first door.

Under your original <img> element, create two new <img> elements and assign them the same src as your original <img> element. Their id values, however, will be assigned door\_b and door\_c respectively. Make sure to place the id before the src.

**12.**

Being a good programmer means doing your best to stay DRY(“Don’t repeat yourself”). Looking at the **style.css** code, it would be repetitive to create CSS ID selectors for #door\_b and #door\_c just to give them the same cursor property.

Navigate back to the **index.html** and assign each image a new class of door-frame. Make sure to place the class between the id and the src.

Listing the id and class in this order is an important reminder of **CSS** specificity (the order by which the browser decides which CSS styles will be displayed).

Hint: An <img> element’s class is defined after the id but before the src:

**13.**

Next, in the **style.css** file, create the .door-frame CSS class selector and assign the following properties and values:

* cursor - pointer
* padding - 10px

Refresh the page after each new CSS property is assigned so you can see how each individual value influences the three doors.

**14.**

Delete the #door\_a ID selector since the .door-frame class selector makes #door\_a redundant.

**15.**

Now you have three doors and the cursor changes on hover for all three; but only your first door opens. Put **JavaScript** to work to open those other two doors!

Go to your **script.js** file and create two new global variables called doorImage2 and doorImage3. Use a **JavaScript** DOM method to assign these global variables to the **HTML** elements with the id of door\_b and door\_c respectively.

**16.**

Under your doorImage1.onclick() arrow function, give doorImage2 and doorImage3 each their own .onclick() arrow functions. These functions will be empty for now.

**17.**

Be mindful to change the logic if you decide to copy and paste the logic from the doorImage1.onclick() function - or else you’ll have three FindFlys!

To prevent this from happening, first create a global variable called oceanDoorPath and assign this URL string as “ocean.png” path.

**18.**

Next, create another global variable called forestDoorPath and assign this URL string as “forest.png” path.

**19.**

Now within the empty .onclick() functions of doorImage2 and doorImage3, write code so that doorImage2 will change to the ocean image and doorImage3 will change to the forest image when clicked.

Now when you refresh the page, click on each door and witness how each closed door opens to something different: a FindFly, a ocean, and outer forest.

**20.**

You have three doors but they’re all huddled in the top left corner. Give your website some flavor and symmetry by expanding your **index.html** and **style.css** pages.

Right after the opening <body> tag, create a new <div> element with the class name "header". Inside this new <div> parent element, create an <h1> child element for the title of the page.

**21.**

In the **style.css** file, under the body selector but above the door-frame selector, create a header CSS selector and assign the background-color the value #00ffff. If you refresh the page, the header is now more visible; but it’s still in the left corner. Assign the text-align property a value of center.

**22.**

Good work - you have your game title. Now it’s time to write the instructions to your game. First, let’s create a title for these instructions and flank this title with a pretty star on each side.

Beneath the <div class="header"> element but above the <div class="door-row"> element, create a new <div> element with the class name "title-row". Inside this element, create two images using star.svg path.

Refresh your page and you should see two small stars huddled in the top-left corner of their <div> underneath the header.

**23.**

Your two stars, without any CSS, are resting in the top-left corner of their <div>. Noticing a trend? Let’s fix this default placement.

Navigate to the **style.css** file and under the .header CSS selector, create a new selector for .title-row. For this selector, assign the following properties with these values:

* margin-top - 21px
* margin-bottom - 21px
* text-align - center

Refresh the page to make sure the stars are centered.

**24.**

Now to add the actual instructions title.

In between the star images, place a <p> sibling element with the class name "instructions-title". Type Instructions inside this <p> element.

**25.**

The text and the stars are centered but the stars still appear to be on different lines. Put them all on the same line.

Create an .instructions-title CSS selector under your .title-row selector and assign the display property a value that will place the title inline with the stars.

Save your code and refresh the page and see how the stars and title are now on the same line.

**26.**

Unfortunately, Instructions is still hard to see. Utilize some CSS to improve your title’s visibility.

Look at your .instructions-title CSS rule again and pair the following properties with their respective values under your display property:

* font-size - 18px
* color - #00ffff
* font-family - ‘Work Sans’

Save your code after each new CSS property is assigned so you can see how each individual value influences your title.

**27.**

With the title now prominently displayed, the next objective is to explain the game to anyone who wants to play. You’ll use an HTML table to display style the instructions. This will make it easier to line up the number labels with the instruction text itself.

In the **index.html** file, beneath the <div class="title-row"> element, create a new <table> with the class name "instructions-row". Inside of that, create two identical empty <tr> elements.

**28.**

Nested inside each <tr> parent element should be two <td> child elements:

* The first <td> of each child pair has the class name "instructions-number".
* The second <td> of each child pair has the class name "instructions-text".

**29.**

In between each <td></td> element with the instructions-number class, add the numbers 1 through 2 so that these two <td></td> elements are numbered sequentially.

**30.**

Going down each <td> element with the "instructions-text" class sequentially, copy and paste the two directions to the game in order:

* Your mission is to open all of the doors without running into the FindFly.
* If you manage to avoid the FindFly until you open the very last door, you win!

Save your code and refresh your page to ensure that your game has two directions listed.

**31.**

While the instructions are now listed, it’s important to improve the spacing and visibility of these instructions.

In the **style.css** file, create the .instructions-row CSS selector under your .instructions-title selector and assign the following properties and values:

* margin - 0 auto
* width - 500px

Refresh the page after each new CSS property is assigned so you can see how each individual value influences your .instructions-row.

**32.**

Underneath that, create the .instructions-number selector and assign the following properties and values:

* padding-right - 25px
* font-family - ‘Work Sans’
* font-size - 36px
* color - #00ffff

Refresh the page after each new CSS property is assigned so you can see how each individual value influences your .instructions-number.

**33.**

Finally, underneath that, create the .instructions-text selector, and assign the following properties and values:

* padding - 10px
* font-family - ‘Work Sans’
* font-size - 14px
* color - #ffffff

Refresh the page after each new CSS property is assigned so you can see how each individual value influences your .instructions-text.

**34.**

Since you’ve centered the instructions, you should also center the doors.

In **style.css**, under the .instructions-text CSS selector, add a .door-row CSS selector and assign the text-align property a value of center.

Save the code and refresh the page to see the three doors centered.

**35.**

The last HTML feature to build is a button that will respond to the status of the game.

Beneath the <door-row> element of your **index.html** page, create a new <div> element with an id of "start" and the class name "start-row". Type Good luck! in between the <div></div> tags.

Refresh the page to see Good luck! appear below the doors on the left.

**36.**

In the **style.css** file, beneath every other CSS selector, create a .start-row CSS selector assign the following properties and values:

* margin - auto
* width - 120px
* height - 43px
* font-family - ‘Work Sans’
* background-color - #eb6536
* padding-top - 18px
* font-size - 18px
* text-align - center
* color - #010165
* margin-bottom - 21px
* cursor - pointer

Save your code and refresh the page after each new CSS property is assigned so you can see how each individual value influences your .start-row.

That was a healthy amount of HTML and CSS additions but your game is beginning to look not only stylish, but also user-friendly, with a logical display of different features.

**37.**

Your game is beginning to look like a game thanks to your HTML structure & CSS manipulations; but what good is a game if you know where the FindFly is always hiding? JavaScript will help solve this problem by helping you randomly generate the door that hides the FindFly.

In your **script.js** file, beneath your global variables, create a randomFindDoorGenerator() function using arrow function syntax.

**38.**

This function will require many new global variables in order for the game’s logic to work.

Create the global variable numClosedDoors and set its value to the amount of doors in the game.

**39.**

Instead of having the FindFly always hide behind the first door, let’s begin crafting the logic to randomly generate the FindFly’s appearance behind any door.

In the randomFindDoorGenerator() function, create a findDoor variable and set its value to a Math method that will randomly generate a whole number between 0 and 2.

To do this, this Math function will utilize:

1. The Math.random() function to generate a decimal between 0 and 1.
2. The numClosedDoors variable as a multiplier with the Math.random() function to generate a decimal value between 0 and 2.
3. The Math.floor() function to round down this decimal value to the closest whole number.

**40.**

Now that your randomFindDoorGenerator() randomly generates one of three possible values (0, 1, or 2), it’s time to write the logic that assigns each of these possible values to a different door where the FindFly could hide.

First, create the global variables openDoor1, openDoor2, and openDoor3 but do not assign any value to them globally. They’ll be assigned values within your randomFindDoorGenerator() function.

Hint: Use let to create global variables that will later be assigned a value.

**41.**

Inside the randomFindDoorGenerator() and directly beneath the Math function, write an if/else if/else statement where each condition is a possible findDoor value of 0, 1, 2.

Hint: Since findDoor is either 0, 1, or 2, the conditions are.

**42.**

Each condition should have a different door holding the FindFly image.

Since there are 3 conditions in this if/else statement, assign the flyDoorPath variable to a different openDoor global variable so that openDoor1 is assigned the flyDoorPath variable under one condition, openDoor2 is assigned the flyDoorPath variable under another condition, and openDoor3 is assigned the flyDoorPath variable in the final condition.

**43.**

Within each of these three blocks of code, under the openDoor being assigned the flyDoorPath variable, assign the other openDoor global variables so that one is assigned the oceanDoorPath variable and the o ther is assigned the forestDoorPath variable.

**44.**

Now look at our three .onclick() functions. In its current state, the logic inside explictly states which image path will replace the original src. Replace these explicit variables from each .onclick() function with an openDoor variable so that the value of doorImage1.src will change to openDoor1, and so on.

**45.**

To see if the randomFindDoorGenerator() is working, you should call it! Write the function call at the very end of the **script.js** file.

With each refresh of the page, check to see if the FindFly appears in a different door. While the gaming logic isn’t fully there yet, you’ve made significant progress towards creating a dynamic webpage that responds to user interactions.

**Building a Winner**

**46.**

You’ll need to create a function called playDoor() that serves two important roles:

1. It decreases the numClosedDoors variable. This is because each time you click a door, the number of available doors to click goes down by one.
2. It checks if the game-winning condition (numClosedDoors === 0) has been met and if so, calls a gameOver() function (which you will write later).

In the **script.js** file, right below your list of global variables, create a new empty function called playDoor().

**47.**

Within this playDoor() function, write code to decrease the numClosedDoors variable every time its called.

Hint: This can be done in several ways—one is to use the decrement operator.

**48.**

Under the code from the previous step, write an if/else statement that determines if the game-winning condition has been reached. If so, call a gameOver() function (again - this gameOver() function hasn’t been created yet).

Hint

The game-winning condition is that the numClosedDoors variable equals 0, so to check that it has been met and call gameOver():

**49.**

The all-important playDoor() function has been written but the question now is where in your **script.js** file should this function be called? If the numClosedDoors variable decreases, that means that a door has been opened. The three door elements’ .onclick() functions are where a door is opened. Inside and at the bottom of each event, call the playDoor() function.

**50.**

One common complaint about poorly-built games is when there’s a flaw in the game’s logic which a player can exploit to win easily. If (numClosedDoors===0) is the winning condition, a player can click the same door (even if it’s opened) multiple times to decrease the numClosedDoors value down to 0 and “cheat” his/her way to victory. You worked hard to build your game - don’t let your players exploit the current logic!

You need logic to make each door clickable only once. Above your playDoor() function, create a new empty function called isClicked() that has door as a parameter.

**51.**

An important global variable storing the source path of the closed door image must be created at this point. It will be very useful for your isClicked() function as well as other JavaScript functions in this game.

Create another global variable called closedDoorPath and assign “door\_closed.png” path as its value.

**52.**

The isClicked() function will return a *boolean* value. You’ll call isClicked() with a door argument and use that door element to make the determination.

Write an if/else statement where the condition checks if the door‘s src is equivalent (or contains) to the closedDoorPath.

If they share the same value, then the door hasn’t been opened yet (meaning it has not been clicked) and you should return false.

Otherwise, the door must be open already (meaning it has been clicked) so the function should instead return true.

**53.**

Now that you’ve written the isClicked() function, put it to use.

Navigate to the three door element .onclick() functions and within each function, wrap the current logic within an if statement to determine if the isClicked() function has *not* yet happened for that particular doorImage.

Adding this logic now protects your game from shortcut victories by making each closed door clickable only once.

**54.**

The time has come to create a gameOver() function so that your playDoor() function can actually end the game when numClosedDoors is equivalent to 0.

Under your door element .onclick() functions, write a new, empty function called gameOver().

**55.**

There’s one global variable that still needs to be created before you can expand the gameOver() function.

Create a startButton global variable and use a JavaScript DOM method to assign its value to the HTML element with the id of start.

Hint: Use the .getElementById() method:

**56.**

Now that you have the startButton variable, let’s expand the gameOver() function.

Add status as a function parameter and write an if statement where the condition checks if status is equivalent to 'win'. If this condition equates to true, then the innerHTML of the startButton should change to You win! Play again?.

Hint

if (status === 'win') {

startButton.innerHTML = 'You win! Play again?';

}

**57.**

Within your playDoor() function, the gameOver() function is called but without passing in an argument. Without an argument, the startButton text will not change when you open all three doors.

Add the string 'win' as the argument for the gameOver() function within your playDoor() function.

After implementing this step, refresh the page and open all three doors and look what happens to your ‘Good luck’ string in the startButton. Your winning condition (numClosedDoors === 0) has been reached. The only problem is that the current logic will always have you “win” as long as all the doors are open, regardless of when you find the FindFly.

**Building a Loser**

**58.**

You need to check if a door has the game-ending FindFly.

Above the isClicked() function, create a new function called isFly() that takes door as its argument. Just like the isClicked() function, isFly() will return a boolean value. Within this new function, write an if/else statement to check if the door.src value is equivalent (or includes) to the flyDoorPath. If they share the same value, that means that particular door has the FindFly and should return true. Otherwise, the isFly() function should return false.

**59.**

You’ve written a function to determine if a door‘s src contains the game-ending FindFly image. Now you must apply this logic into currently existing JavaScript functions.

The playDoor() function now needs a door argument. After the if statement in this function, add an else if condition that checks if the isFly() will equate to true if you pass the door as the isFly() argument.

Hint

else if (isFly(door)) {

// Function code...

}

**60.**

If this isBot() function equates to true, call the gameOver() function with no argument.

**61.**

Since the playDoor() function now needs an argument, look at the door element .onclick() functions. Pass doorImage1, doorImage2, and doorImage3, respectively, as the arguments for the playDoor() function within each .onclick() function.

Hint: playDoor(doorImage1);

**62.**

Now reexamine the gameOver() function. Beneath the current logic, write an else statement that will change the .innerHTML of the startButton variable to Game over! Play again?.

Refresh the page and notice how the text changes to ‘Game Over! Play again?’ if you find the FindFly behind the first door or second door that you open.

Hint

else {

startButton.innerHTML = 'Game over! Play again?'

}

**63.**

There’s a problem with the current gaming logic. Notice that even if you find the FindFly behind the first door or second door that you open, you can still open the remaining doors and override a loss to victory (See what happens to the startButton text once all the doors are open). Adding more logic will fix this game logic flaw.

Add a new global variable named currentlyPlaying and set its value to true.

**64.**

At the bottom of the gameOver() function, set currentlyPlaying to false. You’ll use this value to make sure that additional doors can’t be clicked after the FindFly door is clicked.

Hint: Set currentlyPlaying to false outside of the if/else statement

**65.**

Now looking at the door element .onclick() functions, add to the current if statement a condition that checks whether currentlyPlaying returns true AND that the isClicked(door) function returns false.

Hint: To complete the if condition, you need to use the Boolean AND operator (&&) and the NOT operator (!).

**66.**

Well done—you’ve made it so the game can’t be played once you’ve hit the losing game over condition and the game is now functioning as it should. It lets you know if you’ve opened all the doors to victory and it tells you if you’ve lost because you found the FindFly before all the doors were opened.

There’s one problem though - the only way to reset the values for a new round is to refresh the page. Your next task is to turn your startButton into exactly that - a start button for a new round.

Directly underneath the door element .onclick() functions, add a click handler function for the startButton element. Inside it, call the startRound() function (which hasn’t been written yet).

Hint

To call a function from inside a click handler:

element.onclick = () => {

functionToCall();

}

**67.**

The startRound() function not only has to start a new game; it also has to reset the values from the previous game.

Create this new startRound() function after the .onclick() functions. startRound() should reset the following variables back to their original values:

* All doorImage.src values
* numClosedDoors variable
* startButton.innerHTML variable
* currentlyPlaying variable

The numClosedDoors variable is set to 3, each doorImage.src variable is set to the closedDoorPath variable, the startButton.innerHTML is set to 'Good luck!', and the currentlyPlaying variable is set to *true*.

**68.**

After all these variables are reset, call the randomFindDoorGenerator() function (inside and) at the bottom of the startRound() function.

**69.**

Now that the startRound() function exists, the randomFindDoorGenerator() function at the bottom of the **script.js** needs to be replaced by the startRound() function so that the game’s variables are set to their original values when the game is initially loaded.

**70.**

Currently, the game can reset mid-round if the player clicks on the startRound button. This bug results in every doorImage.src becoming a closed door again before the winning or losing condition is reached.

Wrap the code inside the startButton.onclick() function in an if statement where the condition checks if the currentlyPlaying variable is false so that a player cannot reset the game mid-round.

You could write the condition for this if statement in a couple ways:

if(!currentlyPlaying) {

}

if(currentlyPlaying === false) {

}

Congratulations! You’ve successfully built a full interactive game utilizing your HTML, CSS, and JavaScript knowledge. Be proud of how far you’ve come.