Build a Code Breaker Game With JavaScript

Use JavaScript to accept user input, validation, and run game logic to build a game based on Mastermind.

Watch Intro Video

Table of Contents

Setup

Build

1. 5. Create setHiddenFields Function
2. 6. Make sure the hidden input answer's value is exactly 4 characters long
3. 7. Set the hidden input attempt's value to zero
4. 8. Only set the answer and attempt hidden inputs when they aren't already set
5. 9. Create setMessage function
6. 10. Create validateInput function
7. 11. Call the validateInput function from the guess function
8. 12. Create getResults function
9. 13. Check for correct guess
10. 14. Setup Win Condition
11. 15. Setup Lose Condition
12. 16. Continue Play Condition
13. 17. Create a showAnswer function
14. 18. Create a showReplay function
15. 19. Add showAnswer and showReplay to Win / Lose Conditions

@RajaniCodeNot you?

The Build

Help Me

- Discuss This Project
- Watch Answer Video
- Watch Setup Video
- Start Over

Setup

1. We'll need to know your GitHub username to complete this project.

2. In a new window, head over to GitHub and Fork thecodeschool-projects/CodeBreakerProjectrepo into your account. Once you've done that on GitHub, let us know!

3. Clone down your fork of the repo locally. You can copy and paste either version of this line into your terminal.

4. Open this project's directory in a text editor to complete this project. A text editor like Atom or Sublime Text will do the job. You will make changes to the `src/assets/main.js` file to satisfy the requirements.

   I'm Ready to Start Building

Build

Satisfy the list of requirements below, commit, and push your code to GitHub.

1. ### Create setHiddenFields Function

   Create a function named `setHiddenFields` that sets the `answer` variable equal to a randomly generated whole number between 0 and 9999.

Hint: `Math.random()` can be used to randomly generate a number between 0 and 1 (up to 18 decimal points) and `Math.floor(input)` can be used to round down to the nearest whole number.

2. **Make sure the hidden input answer's value is exactly 4 characters long**

   In our `setHiddenFields` function we need to make sure the hidden input `answer` is exactly 4 characters long. (If our random number generates "42", we want to set the `value` of `answer` to "0042".)

   Hint: In order to add a zero to the front of an answer, it must be a string, not a number. You can convert numbers to strings with `.toString()`. We can create a `while` loop that runs while `answer.length` is less than 4 that puts a `0` before answer's current value.

3. **Set the hidden input attempt's value to zero**

   In our `setHiddenFields` function we should also set the hidden input `attempt` to `0`.

4. **Only set the answer and attempt hidden inputs when they aren't already set**

   Call the `setHiddenFields` function in the body of the `guess` function, but also write some logic so that it's only called when answer and attempt haven't already been set.

   Hint: we can use an `if` condition to only run our code when `answer` or `attempt` is empty (`''`).

5. **Create setMessage function**

   Create a `setMessage` function with one parameter. This function should set the `message` label to whatever is provided to the parameter.

   Hint: With a label, you'll want to set its `.innerHTML`, not its `.value`.

6. **Create validateInput function**

   Create a function `validateInput` with one parameter. If the parameter has a `length` of 4 return `true`, otherwise use the `setMessage` function to set the `message` label to `"Guesses must be exactly 4 characters long."` then return `false`

7. **Call the validateInput function from the guess function**

   Create an `if` condition block that uses `validateInput` with a parameter of `input.value` as the conditional. If `validateInput` returns `false`, then use `return false` to stop execution of the `guess` function, otherwise we should increment the `attempt` hidden input by 1.

   Hint: You can negate a value on the `if` statement by using the exclamation point, like this: `if(!someValue)`.

8. **Create getResults function**

   Create a `getResults` function that has one parameter. In this function, we need to add the results of the user's guess to our `results` div's `innerHTML`. Each result should begin with `<div class="row"><span class="col-md-6">' + input + '</span><div class="col-md-6">` where `input` is the value the user guessed. Then for each character, you should add `<span class="glyphicon glyphicon-ok"></span>` if the character is in the correct position in the `answer`, a `<span class="glyphicon glyphicon-transfer"></span>`

if the character is in the `answer` but isn't in the right position, and `<span class="glyphicon glyphicon-remove"></span>` if the number isn't in the `answer` at all. Don't forget to close your divs!

Hint: You can create a variable to hold the initial div, then add each character's results to that variable in a `for` loop, then add the closing `div` tags after the loop. After which you can just set the `results` element's `innerHTML` to that variable.

9. ## Check for correct guess

   In our `getResults` function create a variable that counts how many characters were guessed correctly, if all characters were guessed correctly the function should return `true` otherwise `false`

10. ## Setup Win Condition

    Add a call to the `getResults` function at the end of our `guess` function. If `getResults` returns true use the `setMessage` function to set the `message` label to `"You Win! :)"`.

11. ## Setup Lose Condition

    If `getResults` returns `false` and the hidden input `attempt` value is greater than or equal to 10 use the `setMessage` function to set the `message` label to `"You Lose! :("`.

12. ## Continue Play Condition

    If neither a win or lose condition is met use the `setMessage` function to set the `message` label to `"Incorrect, try again."`.

13. ## Create a showAnswer function

    Create a function `showAnswer` that has one parameter. This function should set the `innerHTML` of the `code` label to the `value` of the `answer` hidden input. In addition to this it should take the parameter as a `true` or `false` (indicating if the player won or lost) if the parameter is `true` add `success` to `code`'s `className` otherwise it should add `failure`. (note the space before `success` and `failure`)

14. ## Create a showReplay function

    Create a function `showReplay` with no parameters. This function will change the `style.display` of `guessing-div` div to `none` and the `style.display` of the `replay-div` div to `block` making it so the user can start over after they win or lose the game.

15. ## Add showAnswer and showReplay to Win / Lose Conditions

    When a player wins in addition to `setMessage` call, they should also call `showAnswer` passing `true` for it's parameter, and finally make a call to `showReplay`. When the player loses they should call `showAnswer` with `false` for the parameter and then `showReplay`.

When you're ready, commit your code and push it to GitHub. We'll checkout your repository and validate all of the build tasks are done.

Check My Work on GitHub

If you're having trouble, you can watch theAnswer Videoto see one way of completing this project.

Close

How to Fork

How to Fork modal!
Close

Close

Close

Close

Create a Bookmark

In order to bookmark this content, we'll need to add it to a Code School account.

Create a Free Account

Already have an account?

Sign in