REVIEW ALGORITHMIC THINKING BY BUILDING A DICE GAME

Introduction:

Algorithmic thinking involves the ability to break down complex problems into a sequence of well-defined, step-by-step instructions.

In this Dice game project, you’ll learn how to manage game state, implement game logic for rolling dice, keeping score, and applying rules for various combinations.

This project covers concepts such as event handling, array manipulation, conditional logic, and updating the user interface dynamically based on game state.

Step 1:

In this project, you will learn algorithmic thinking by building a dice game. There are a total of 6 rounds and for each round, the player can roll the dice up to 3 times and collect a score.

The HTML and CSS have been provided for you. Feel free to explore them. When you are ready, you will need to set up your HTML variables. Get all of your `.die` elements and assign them to a `listOfAllDice` variable. Get your score inputs (the `input` elements in your `#score-options`) and score spans, and assign them to `scoreInputs` and `scoreSpans`. Assign the `#current-round` element to `roundElement` and the `#current-round-rolls` element to `rollsElement`, then do the same for your `#total-score` and `#score-history` elements. Assign your `#roll-dice-btn`, `#keep-score-btn`, `#rules-btn`, and `.rules-container` to variables with properly formatted names.

When the user clicks on the `Show rules` button, they should be able to toggle between showing and hiding the game rules. Create a variable `isModalShowing` to track the state of the game rules.

Each time the user rolls the dice, you will need to keep track of all of the dice values. Create a variable `diceValuesArr` to track this.

Throughout the game, you will need to keep track of the current score, total score, number of rolls and which round the player is on. Declare `rolls`, `score`, and `round` variables for this purpose.

Think about what the starting value of each of these variables should be. All of these variables should be able to be reassigned.

Step 2:

When the user clicks on the `Show rules` button, the rules for the game should display on the screen. When they click on the button again, the rules should be hidden.

Use an event listener to invert the value of the `isModalShowing` variable, toggle the visibility of the `rulesContainer`, and change the text of the `rulesBtn` to `Show rules` or `Hide rules`.

Step 3:

When the user clicks on the `Roll the dice` button, five random die numbers should be generated and displayed on the screen.

Build out the logic such that clicking on the `rollDiceBtn` generates five random numbers between 1 and 6 inclusive, sets the `diceValuesArr` to contain only those five numbers, and displays the numbers in order in the `listOfAllDice` elements.

Step 4:

For each round in the game, users are allowed to roll the dice a maximum of three times. If a user clicks the `rollDiceBtn` but has already made three rolls, the browser should show an `alert()` to indicate they must select a score - otherwise, it should roll the dice as it currently does and increment the `rolls` variable.

Step 5:

You'll need to be able to update your `rolls` and your `round` on the page. Create an `updateStats` function to update the text of those two elements with the appropriate values. Then, call that function when your `rollDiceBtn` is clicked and the dice are rolled.

Step 6:

Each time you roll the dice, you could end up with a `Three of a kind`, `Four of a kind`, `Full house`, `Straight` or a random combination of numbers. Based on the outcome, you can make a selection and add points to your score.

Start by creating a function called `updateRadioOption` that takes an index and a score value as arguments. It should set the `scoreInputs` at that index to be enabled, set the value of that input to the score, and display `, score = ${score}` in the correct `scoreSpans` element.

Step 7:

If you roll the dice and get a `Three of a kind` or `Four of a kind`, then you can get a score totalling the sum of all five dice values. To calculate this, create a `getHighestDuplicates` function which takes an array of numbers. The function will need to count how many times each number is found in the array.

If a number appears four or more times, you will need to update the `Four of a Kind` option with your `updateRadioOption` function. If a number appears three or more times, you will need to update the `Three of a Kind` option. In both cases, the score value should be the sum of all five dice.

Regardless of the outcome, the final option should be updated with a score of `0`. Make sure to call your `getHighestDuplicates` when the dice are rolled.

Step 8:

Before each dice roll, you will need to reset the values for the score `inputs` and `spans` so a new value can be displayed.

Create a `resetRadioOptions` function. Your function should iterate through the `scoreInputs` to disable them and remove the `checked` attribute. Your function should also remove the text from each of the `scoreSpans`. Finally, call this function before you roll the dice.

Step 9:

When you roll the dice and make a selection, you are not able to keep the score you selected and move onto the next round.

Create an `updateScore` function to add this functionality. Your function will need two parameters for the user selected score option. The first parameter will be passed the `value` of the radio button, remember this is a string, and the second parameter will be passed the `id` value of the radio button, which is the type of score they achieved.

The function will need to add the user selected value to the score, update the total score text on the page, and add a new `li` element to the score history `ul` element, using the format `${achieved} : ${selectedValue}` for the `li` element content.

Step 10:

After a user makes a selection, they should be able to keep that score and move onto the next round by clicking the `keepScoreBtn`.

When that button is clicked, you should find which radio option is checked and capture its value and id attributes. If the user has selected an option, call your functions to update the score, reset the radio options, and add the value and id to the score history.

If the user has not selected an option, display an alert informing them to do so.

Step 11:

At this point in the game, you are able to roll the dice, make a selection and play for a few rounds. However, you should notice that there is no end to the game and there are infinite rounds. According to the rules, there should be a total of six rounds and then the game ends with the final score.

After running your logic when the user selects a score, you should check if 6 rounds have been played. If so, display an alert with the user's final score after 500 milliseconds.

Step 12:

If you go through six rounds of the game, you should see the `alert` show up with your final score. But when you dismiss the `alert`, you are able to keep playing for more rounds past the original six. To fix this, you will need to reset the game.

Declare a `resetGame` function to do so. Reset all of the `listOfAllDice` elements to display `0`, update `score` and `rolls` to be `0`, update `round` to be `1`, set the `totalScoreElement` text to the user's total score, clear the score history by setting it to an empty string, set the `rollsElement` text to the number of rolls, and set the `roundElement` text to the current round. Finally, reset all of the radio buttons to their initial states.

Call this function after displaying the `alert` with the final score.

Step 13:

If the user rolls three of one number, and two of another number, this is called a full house. Declare a `detectFullHouse` function that accepts a single argument. The function will be passed the `diceValuesArr` array when called.

Your `detectFullHouse` function should check if the user has rolled three of one number and two of another number. If so, it should update the third radio button to display a score of `25`, with the correct attributes. Regardless, it should always update the last radio button to display a score of 0, with the correct attributes.

Don't forget to call your new function when the dice are rolled.

Step 14:

For the last portion of the game, you will need to create an algorithm that checks for the presence of a straight. A small straight is when four of the dice have consecutive values in any order (Ex. in a roll of `41423`, we have `1234`) resulting in a score of `30` points. A large straight is when all five dice have consecutive values in any order (Ex. in a roll of `35124`, we have `12345`) resulting in a score of `40` points.

Declare a `checkForStraights` function which accepts an array of numbers. If the user gets a large straight, update the fifth radio button with a score of `40`. If the user gets a small straight, update the fourth radio button with a score of `30`. If the user gets no straight, update the last radio button to display `0`.

Call your `checkForStraights` function when the `rollDiceBtn` is clicked to complete your dice game!