I will begin this documentation with a short description of my naming conventions for ease of reading. Thus, we have the three given requirements for a password to be strong:

- Its length is between 6 and 20 characters: **First Constraint**
- It has to have at least one uppercase letter, one lowercase one and one digit: **Second Constraint**
- It cannot contain three repeating characters in a row: **Third Constraint**

Also, since it wasn't specifically stated in the text of the problem, I have created an algorithm that computes the minimum number of changes for the password. It will not however elaborate on what those changes should be.

I used a number of functions to check these constraints, and in the following I will explain them.

First Constraint

The method "HowManyCharactersToSize" takes a string as an argument representing the password and has three possible return values that are the number of changes required for the password to fulfil the first constraint. If it already fulfils the constraint, it will return 0. If the password is too short, it will return the result of subtracting the length from 6, or, in other words, a positive number. Finally, if the password is too long, the return value is a negative number, that is, 20 minus the size of the password.

Second Constraint

Since this particular requirement can be divided into three smaller ones, I have done just that with three methods that work basically in the same way, all of them receiving a string and returning either 0 or 1, based on whether or not a specific character can be found in the string. On a more abstract note, the combined return values of these functions represent how many changes are required for the password to fulfil the Second Constraint.

To give a quick overview of the functions(EnoughUpperCase, EnoughLowerCase, EnoughDigits), they search character by character in the input string for a(n) uppercase letter/lowercase letter/digit and, if one was found, 0 is returned. Otherwise, return 1.

Third Constraint

For this one the algorithm is similar to the one I used for the Second Constraint, and by that I mean that, going one character at a time, I searched for groups of three by checking for each character if it is the beginning of such a group (that is, if it has the same value as the next one and the next one after that). If found, a counter is increased. At the end, the counter is returned.

Minimum number of changes

Now, I have noticed that the problem can be viewed from three distinct cases based on how the First Constraint is broken/respected, and by that I mean whether HowManyCharactersToSize returns a positive number, a negative one or simply 0. Furthermore, since the goal is to minimize the amount of changes and since by fulfilling the Second Constraint one may also fulfil the Third, I have taken the maximum out of the number of needed changes for each of these two constraints (a sort of a "kill two birds with one stone" approach).

If the password has enough characters, that is, HowManyCharactersToSize returns 0 for that password, all that is required to do are edits that will not affect its length. As such, the number of edits is dictated by the other two constraints and is, as stated above, the maximum between the numbers of changes for the two.

If the size of the password happens to be greater than 20, one must remove some characters on top of editing/adding other characters for the Second and the Third Constraints. Since the removes must happen, the final number of changes in this case is the number of removes necessary (i.e. the absolute value of HowManyCharactersToSize when called for the password) plus, yet again, the maximum described above.

Finally, if the password's length is less than 6 (meaning some additions have to take place). Since one must add some characters to satisfy the First Constraint, one may as well add those that will fulfil the Second and the Third as well (adding characters would interrupt groups of three identical characters). Thus, in this case, the minimal number of changes can be obtained by taking the maximum from the amount of changes needed for all three constraints.