COGNISESS SOFTWARE ASSESSMENT

Thank you for giving such an interesting task as a part of the assessment for this placement opportunity at your company. This is the first time I have worked on .Net with C#. It was a good and challenging experience to me and I hope this has helped me to get familiarised with the type of work that I would be learning if given a chance to work at Cognisess.

My submission includes 3 files; In the first file “program.cs” I have done the backend coding on .Net with C# by generating random numbers with changing difficulties and increasing the score whenever user enters the right number. But I was facing problems in calling C# function in JavaScript. I tried using AJAX, Fetch() by creating local server but I couldn’t get the desired output which is why I did the second coding in a slightly different way to come up with the required solution(“numerical memory.html”). I also tried using revoke in .Net Core with Web Applications (Razor); I understood the method but couldn’t implement in my code as my laptop didn’t generate a .razor file.

I took 6 hours to do the task which is in the second file whereas approximately 3 hours to do the first one.

I think the way this application increases difficulty by increasing the number of digits appearing on the screen justifies the fact that human brain can only remember 7-8 digit numbers if seen for such a small time. If the user enters a wrong value then it doesn’t increase the difficulty and gives the random number having same number of digits. Whereas with every correct answer this application adds a digit to the random number thereby increasing its difficulty. One more way of increasing difficulty is to avoid any sort of patterns that can potentially occur while randomising. It can be done by training the app to shuffle the digits after detecting a pattern in the “generated random number”. In general, computer generated random numbers aren’t actually random. They can follow subtle patterns that is quite hard to observe and might take a long time to get detected. Example- a simple random number generator could be built by timing the intervals between a user’s keystrokes. There can be correlations and patterns in these timing that makes random numbers less random. To avoid such type of things we can use a two source extractor algorithm that can eliminate the patterns or the correlations. Since it is a short (60 seconds) numerical memory test, we wouldn’t want to go into the complexity of random number generation.

If I had more time, I would make it more presentable and appealing. Not only this, but I would have introduced the visual representation (a line graph having question number on x axis and response time on y axis) of the response time taken by the candidate to answer each question. Apart from this, I would have given more emphasis on making a successful bridge between the C# and JavaScript.

SUBMISSION BY: SIDDHARTH SHARMA