Premature optimization is known as how one makes code or design more complex and less readable for the sake of having better performance without justifying a reason for faster performance need such as an actual measure and comparison against the project’s goal or objective. 1

Premature optimization is often redundant in the initial phase as it does not add value to the program and does not make the programme run faster as there is no sign of proven justification. Some setbacks associated with premature optimization could possibly cause programmers to waste lots of resources such as time, money and effort to build the fastest program without even acquiring a proper programme. Such efforts would overly complicate the program and increase the likelihood of creating more sophisticated problems at an early stage.2 Time is wasted as programmers are unsure where the bottleneck of their codes is and they would estimate the efficiency of each respective code by blindly guessing. With modern technology, operations are not CPU-bound and therefore turning application codes will cause the operation to wait faster.1 As a result, time and resources are drained to optimise fast code and it does not help to improve the programme significantly.

Other than premature optimization, one can focus on writing clear and readable codes which are easier to write correctly, understand, refactor and optimize at a later stage when it is necessary. One can demand proof when there are requests for premature program optimization as optimization goals followed by measurements are to be achieved before the execution of premature optimization. 1

However, premature pessimization should be avoided at all cost as the consequences are more undesirable.3 Premature pessimization is defined as writing complex doses by taking in unnecessary information thus causing inefficiency without reasons. Codes with equivalent complexity that flow out of our fingertips naturally would run the programme much faster. Also, avoiding it does not mean that efficiency will be affected.4

Moving on, the difference between avoiding premature optimization and premature pessimization is that optimizing code early for faster performance can be a waste of time as it often leads to code that is difficult to read, maintain or modify and should be done at the final phase. However, we can avoid it by taking advantages of modern compilers and processor architecture to generate faster codes before starting the optimization process.5

For instance, premature pessimization is defining pass-by-value parameters, using postfix, assignment inside constructors and etc whereas, avoiding premature pessimization are to pass by reference, to prefer calling prefix operators and the use of code idioms that naturally flows out from our fingertips.1

From my personal experiences of developing a game in GAM100 and GAM150, after finishing a specific feature, one common beginner’s mistake I often make is premature optimization by rewriting new codes that are more complex, shorter and faster but to realise that lots of time and efforts are wasted on minor changes and could be better applied on implementing other additional features. Moreover, I was taught by professors to avoid premature pessimization such as calling prefix operators for faster performance even though both produce the same results as compared to calling postfix and only do it during certain situations.

# References

1. (Sutter, 2004)

2. (Effectiviology, 2020)

3. (Krajewski, 2019)

4. (Elegant Chaos Limited, 2014)

5. (Mutz, 2016)