CGP3011M

Game Engine Architectures

Gabrielle Watson

13420435

Design

The logging system has functions to activate and deactivate logging system. The system must be activated before the any logs will be recorded, and must be deactivated to avoid memory leaks. This isn’t ideal, but it does allow for more organized printing as will be explained later. After being activated, you can call a function to create log, which will take any number of arguments, but first it must take a tag and the format you wish for the log to be printed in. The tags themselves are debug, error, performance, and miscellaneous which should cover most scenarios. By taking in any type and number of arguments it should reduce the amount of work needed to use the system. Instead of taking the time to convert any information you wish to be logged into a character arrays and then either call the log function multiple times or combine them, the function only has to be called once and the information can be passed through as it is. This is much quicker despite having to also enter the format.

When you wish to print the logs another function must be called. When the function is called you state which logs you wish to be printed, and whether the logs are to be printed to the console or to a text file. You choose which logs are to be printed by choosing which tag(s) you want printed. This seemed to be the most useful way of organizing the logs, and will allow the user more choice than simply printing all of the logs (all logs can still be printed by printing all tags). When the logs are then printed, they are organized into the different tags. This should make it easier to read by grouping similar information together, such as keeping all performance logs together. Of course this is less helpful when there are only a few logs, and could even make it more difficult to read. Additionally, there are only a few tags to choose from, so there is still the chance the logs will be dumped together in the miscellaneous category. If there were any more tags though the logs could be spread to an unnecessary level.

If the logs are being printed to the console window then calling the print function multiple times will print the new after the previous logs. However, if printing to a text file every time the print functions is called it will overwrite anything currently in the file. The intention behind this design is that all the logs will be printed at once to make reading through the data easier, but still allow someone to watch a value change over time by either printing to the console or logging the variable over numerous iterations before printing. Though this does mean that to keep the logs they must be stored elsewhere which requires much more effort from the user, but you could argue that deleting the logs would also take this amount of effort so it was decided this was an acceptable compromise. Storing the logs is made easier by the time and date also being printed along with the logs themselves.

The system will only work if running in debug mode. This means that when running a release build it has no impact on the performance for the end user, and means the developers do not have to actively remove the code themselves which is time consuming.

Implementation

The logs are stored as a struct that forms a linked list. The structs hold the message that is to be printed, the tag, and the pointer to the next node in the list. Storing the logs does mean having the impact on performance that comes from maintaining the list, but does allow for more user friendly printing options. Plus if the user is printing to a text file it means only opening the file once during printing, whereas if the logs were being printed immediately the file would be opened and closed many times having an impact on performance as well. The tagging system takes advantage of an enum to limit the options the user has.

The message for the log can be any number of arguments due to the use of a variadic argument list. The list in created using the arguments passed into the logging function, and then the list is traversed through and each argument is appended to the message character array in the log. The variadic argument list also allows the user to pass in any type of argument because the function used to “print” the arguments to the message converts all arguments to strings.

No logging will occur unless it is activated using a function. This function also creates the root node of the linked list and the node used to traverse the list. This means the logging system must be deactivated using another function which deletes these nodes and frees the memory they were using. Otherwise there would be a memory leak. It also deletes the rest of the list in case the printing function hasn’t been called yet (as will be explained below). It was decided the root and traverser nodes would be created in the activating function because the other options would be having to call another function to do this (unnecessary effort for the user) or every time a log is created it would check to see if these nodes had been created and create them if necessary. This may mean that two extra nodes would be created the first time a log is made, probably while the game is running as opposed to when the game is started like it is currently, which is more likely to affect performance at a crucial moment. Additionally, activate and deactivate functions were already required so no unnecessary function were being made.

If the game is being ran as a release build the logging system will not be activated so it will not have an impact of game performance. By not allowing the system to be activated it also means the user does not have to remove any logging code before running a release build. This is achieved using macros to check whether the game is running in debug mode.

Finally, when printing the logs the function uses Booleans to check which type (defined with the tag system) of logs the user wishes to print, and whether it should print to the console or to a predefined text file. It will then loop through the linked list to find logs of that type and print them as one sections. This does mean that if more than one type is being printed then the list must be traversed through multiple times. Unfortunately this has a significant impact on performance which is increased the more logs there are and the more types are being printed. An alternative to this would be sorting the list before printing, but this has the potential to have a larger impact if there are only a small amount of logs or types chosen to print. Given that the impact to performance is unlikely to happen during the time logging is taking place, and so will have no impact on the data being logged, this was deemed acceptable. Once the logs have been printed they are deleted from the list and the memory freed, preparing the list for the next set of logs while keeping the length in check.