Requirements for Logging - S1

By group 26 - The Fishermen

In assignment 1 we are required to list the requirements for the logging of our game Fish.io. We haven chosen to do this in a separate document, as it would be easier to distinguish the different assignments.

The requirements are listed in the same way as our game, so we will use the MoSCoW method. Firstly, the functional requirements of the logging are listed as must, should, could and won't haves. Secondly, the non-functional requirements are listed.

Functional Requirements

1.1 Must haves

- The game Fish.io must be able to output a text file containing logs. This must happen while the game is still running.
- There must be several different *log levels* to specify different levels of necessity. The level a developer is interested in can be set in the code of the project. The four different kinds of levels that must be implemented are:
    - ERROR: A fatal error, in other words, a crash or something that must not have happened acquired.
    - WARNING: Something unexpected happened. but execution can continue.
    - INFO: Something with significance happened. Examples: A new game has started the player has died or state transitions.
    - DEBUG: Something normal happened. Examples: Enemy fish is spawned, eaten.
    - The order of importance is ERROR>WARNING>INFO>DEBUG.
- The logging must be implemented in a standardized way.
    - The logging will follow the same implementation of the java logging API(but will not make use of it). See 1.1, first diagram at: https://docs.oracle.com/javase/8/docs/technotes/guides/logging/overview.html
    - This means that the application can make a logging request to the logger. The logger class decides if it logs the event based on the log level, using the filter. If this comparison succeeds, the log event is passed to the handler class.
    - The handler class uses a formatter or filter when it has to output the logs to a text file. It then outputs the logs.
- The logging must be printed out as [LOGLEVEL] : - <LOGMESSAGE>

1.2 Should haves

- The logging should have timestamps. These timestamps should be displayed before the log message as [HOUR:MIN:SECONDS] [LOGLEVEL] : - <LOGMESSAGE>

- These timestamps can be turned on and off.
- The handler should have a custom formatter.
- The logging should output to the console too. This could be an option.

1.3 Could haves

- There could be an extra log level, called TRACE, which records all user input, such as clicks and keyboard input. This log level could also log speed updating the player fish and enemy fish.
    - The order of importance is ERROR>WARNING>INFO>DEBUG>TRACE.
- The logging could be included as a text window/console, which could be enable in the options menu.
    - This could include option on which log levels to show in this window.
    - The window could be separate from the main game and resizable so that both the game and the log window can be opened at the same time.
- The logging could also return the class and method from which the logger was called.

1.4 Won't haves

- The logging system will not be implemented like an Aspect Oriented programming paradigm as this is too invasive and too convoluted for a *basic* logging system .
- The logging system will not implement the log levels from the logging API as these are too complex for a project this size.

Non-functional requirements

- The logging shall work in Windows (7 or higher) and Linux operating systems.
- The development process will take place according to the scrum method.
- The logging shall be written and executed in Java.
- A first, fully working, version of the logging shall be ready and submitted to the staff of the course on the 18th of september.
- The logging classes will have to adhere to the same non-functional requirements of Fish.io itself, with exception of the modified non-functional requirements in this document.