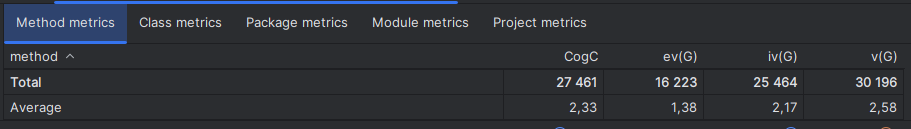
José Trigueiro 58119

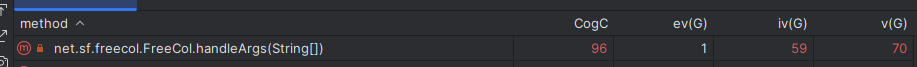
**Code Metric: Complexity Metrics**

The Complexity Metric is the level of difficulty in understanding and maintaining a piece of software code. It measures how much effort is required to comprehend the code, make modifications or fix bugs.

* **Method metrics:**

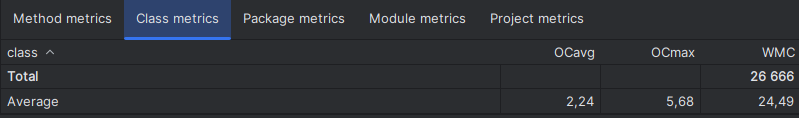


* **CogC** (Cognitive Complexity): is a measure of how difficult a unit of code is to intuitively understand;
* **ev(G)** (Essential Cyclomatic Complexity): is a version of Cyclomatic Complexity that ignores structures that allow multiple statements to be sequentially executed as a group. It provides a measure of the number of decision points in a method, plus one for the method entry;
* **iv(G)** (Design Complexity): it’s a measure of the intricacy of the structure of a software system, the metric quantifies the level of difficulty in managing and evolving a particular design of software, for example a high design complexity often leads to a higher likelihood of errors, as it can be more difficult to understand the interaction between different components of the software;
* **v(G)** (Cyclomatic Complexity): it indicates the complexity of a program and the quantitative measure of the number of linearly independent paths through a program’s source code.

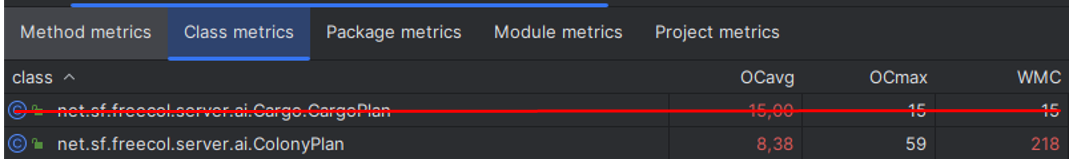


The code smell, Large Method I found on “handleArgs()” relates to the fact that it has a large impact on the average value, due to its size and complexity of understanding, affecting heavily its Cognitive Complexity, Design Complexity and Cyclomatic Complexity metrics.

* **Class metrics:**

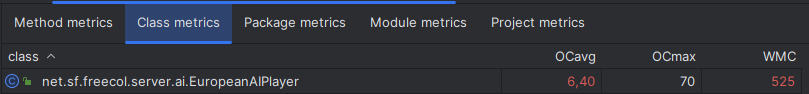


* **OCavg** (Average Operation Complexity): it measures the average complexity of operations (methods) in a class. It’s calculated by summing the complexities(for ex. Cyclomatic Complexity) of all operations in the class and dividing by the number of operations. A high Average Operation Complexity might indicate that the operations in the class are doing too much and could be broken down into smaller, more manageable methods. This can make the code easier to understand, test, and maintain;
* **OCmax** (Maximum Operation Complexity): it measures the complexity of the most complex operation (method) in a class. It’s calculated by determining the complexities of all operations in the class and selecting the highest one. A high Maximum Operation Complexity might indicate that there is a method in the class that is doing too much and could be broken down into smaller, more manageable methods. This can make the code easier to understand, test, and maintain;
* **WMC** (Weighted Method Complexity): it measures the total complexity of all methods in a class. It’s calculated by summing the complexities(for example Cyclomatic Complexity) of all methods in the class. A high WMC indicates that a class may be doing too much and could be broken down into smaller, more manageable classes. It also suggests that the class may be harder to maintain and more prone to errors.

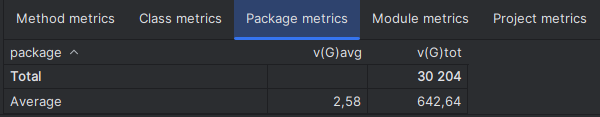


The code smell, Large Class I found on “ColonyPlan.java” relates to the fact that it has a large impact on the average value, due to its size and complexity of understanding, affecting heavily its Average Operation Complexity, Maximum Operation Complexity and Cyclomatic Complexity metrics.

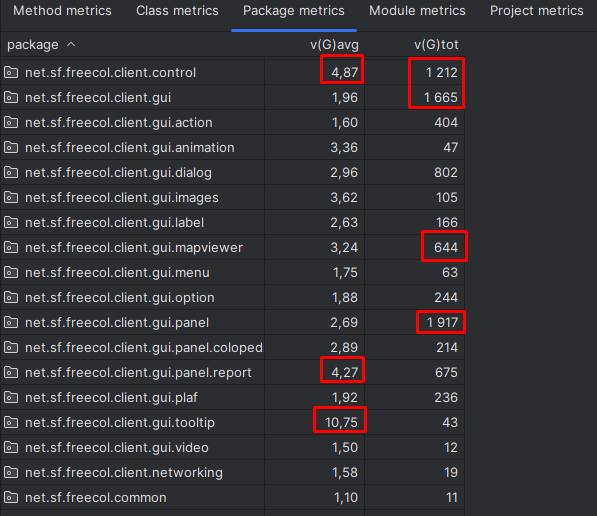
The code smell, Message Chains found in the methods isAggressive()” & “isLikesAttackingNatives()” & “needsMoreDragoons()” & “reallyNeedsMoreDragoons()” & “reallyNeedsMoreArtillery()” & “needsMoreArtillery()” in “EuropeanAIPlayer.java” also affect the same metrics.



* **Package metrics:**

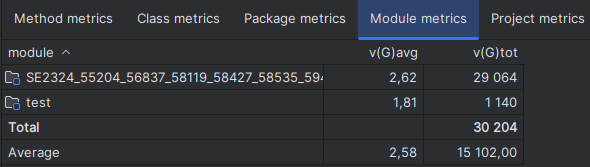


* **v(G)avg** (Average Cyclomatic Complexity): it's a metric that measures the average complexity of all the classes in a package and is calculated by summing the complexities of all classes in the package and dividing by the number of classes. A high Average Cyclomatic Complexity might indicate that the classes in the package are doing too much and could be broken down into smaller, more manageable classes. This can make the code easier to understand, test, and maintain;
* **v(G)tot** (Total Cyclomatic Complexity): it measures the sum of the cyclomatic complexity of all the classes in a package. It’s calculated by summing the cyclomatic complexities of all classes in the package. A high Total Cyclomatic Complexity might indicate that the classes in the package are doing too much and could be broken down into smaller, more manageable classes. This can make the code easier to understand, test, and maintain.

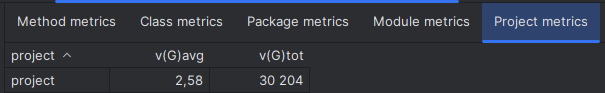


As for bad package metrics we have a mix of bad values all throughout the packages compared to the average but it doesn’t seem that bad.

* **Module metrics:**



* **v(G)avg** (Average Cyclomatic Complexity): it’s a metric that measures the average complexity of all the classes in a module. It’s calculated by summing the complexities of all classes in the module and dividing by the number of classes. A high Average Cyclomatic Complexity might indicate that the classes in the module are doing too much and could be broken down into smaller, more manageable classes. This can make the code easier to understand, test, and maintain;
* **v(G)tot** (Total Cyclomatic Complexity): it measures the sum of the cyclomatic complexity of all the classes in a module. It’s calculated by summing the cyclomatic complexities of all classes in the module. A high Total Cyclomatic Complexity might indicate that the classes in the module are doing too much and could be broken down into smaller, more manageable classes. This can make the code easier to understand, test, and maintain.
* **Project Metrics:**



* **v(G)avg** (Average Cyclomatic Complexity): it’s a metric that measures the average complexity of all the modules in a project. It’s calculated by summing the complexities of all modules in the project and dividing by the number of modules. A high Average Cyclomatic Complexity might indicate that the modules in the project are doing too much and could be broken down into smaller, more manageable modules. This can make the code easier to understand, test, and maintain;
* **v(G)tot** (Total Cyclomatic Complexity): it measures the sum of the cyclomatic complexity of all the modules in a project. It’s calculated by summing the cyclomatic complexities of all modules in the project. A high Total Cyclomatic Complexity might indicate that the modules in the project are doing too much and could be broken down into smaller, more manageable modules. This can make the code easier to understand, test, and maintain.