QA-Konzept und Durchführung

Grundsätzlich sollte sich das Team an folgenden Punkten, die zu nicht-funktionalen Qualitätsmerkmalen gehören orientieren:

- Verstehen meine Teammitglieder diesen Code mithilfe des JavaDocs und meiner Dokumentation?
- Hält sich der Code an den vereinbarten Programmierstil. Dabei haben wir uns für den Google Java Style Guide entschieden. Ausserdem wurde Checkstyle im Gradle konfiguriert, womit man sein Code selber testen kann.
- Hat es bereits Code mit ähnlicher Funktionalität an anderer Stelle?
- Würde ich diesen Code an dieser Stelle im Projekt erwarten?
- Lässt sich die Lesbarkeit dieses Codes erhöhen?
- Könnte ich diesen Code ändern oder ist der Code übertragbar?

JavaDoc

Durch das Dokumentieren und der Zusammenfassung der Methoden und Klassen wird der Aufwand für Teammitglieder, die Methoden und Klassen zu verstehen und weiter zu bearbeiten geringgehalten. Für die Dokumentation der Methoden und Classen sind die Personen, die die Klassen und Methoden schreiben zuständig. Um die Dokumentation zu vereinfachen wird der Plugin JavaDoc in IntelliJ verwendet.

Wir sind der Meinung, dass unser Code recht gut dokumentiert wurde, der Code war für alle Gruppemitglieder, dank der detaillierten Dokumentation einfach zu verstehen. Wir müssen allerdings gestehen, dass der Code sich nicht immer an den vereinbarten Programmierstil hielt, was allerdings mit der Zeit immer besser wurde.

Logging

Um Fehlersuche zu vereinfachen und zu beheben werden Logger verwendet. Die Idee dahinter ist, dass am Anfang jeder Aktion und am Ende jeder Aktion eine Lognachricht verschickt wird, wodurch man die Fehler einfacher hervorheben kann. Ausserdem werden die Lognachrichten in einem Logfile gespeichert. Wir haben uns für die SLF4J API mit logback-classic und core Konfiguration entschieden.

Wir haben Loggers verwendet, um isolierte Codestücke in unserem Projekt zu überprüfen.

```
14:38:20.161 [Thread-1] INFO  server.Client - clientreaderThread started
14:38:33.337 [Thread-1] INFO  server.Client - changing the name
14:39:05.595 [main] INFO  Start - ClientThread started
14:39:05.614 [Thread-0] INFO  server.Server - connected to Client
14:39:05.614 [Thread-0] INFO  server.Server - get input and output stream
14:39:05.614 [Thread-1] INFO  server.Client - Socket created
14:39:05.617 [Thread-0] INFO  server.Server - ServerThreadForClient started
14:39:09.671 [Thread-1] INFO  server.Client - clientreaderThread started
14:39:16.327 [Thread-1] INFO  server.Client - changing the name
14:39:28.330 [Thread-1] INFO  server.Client - Quitting
14:40:18.393 [Thread-1] INFO  server.Client - Quitting
```

Code Coverage

Unser Ziel war Unit Test durchzuführen, um die Kernkomponente unseres Spieles zu testen. Zentral wurde dabei auf die Gamelogik und Client-Server Interaktion geachtet, was wiederum auch Encoding-Decoding des Protokolls und Lobby-Player-Management beinhaltet. Als Tool wurde dabei das Plugin JaCoCo benutzt. Da wir bei Meilenstein 4 noch nicht alle relevanten Eigenschaften der Kernkomponenten getestet haben, war es unserer Meinung nach sinnvoller, die Code Coverage Messungen erst bei Meilenstein 5 durchzuführen.

Metrik-Messungen

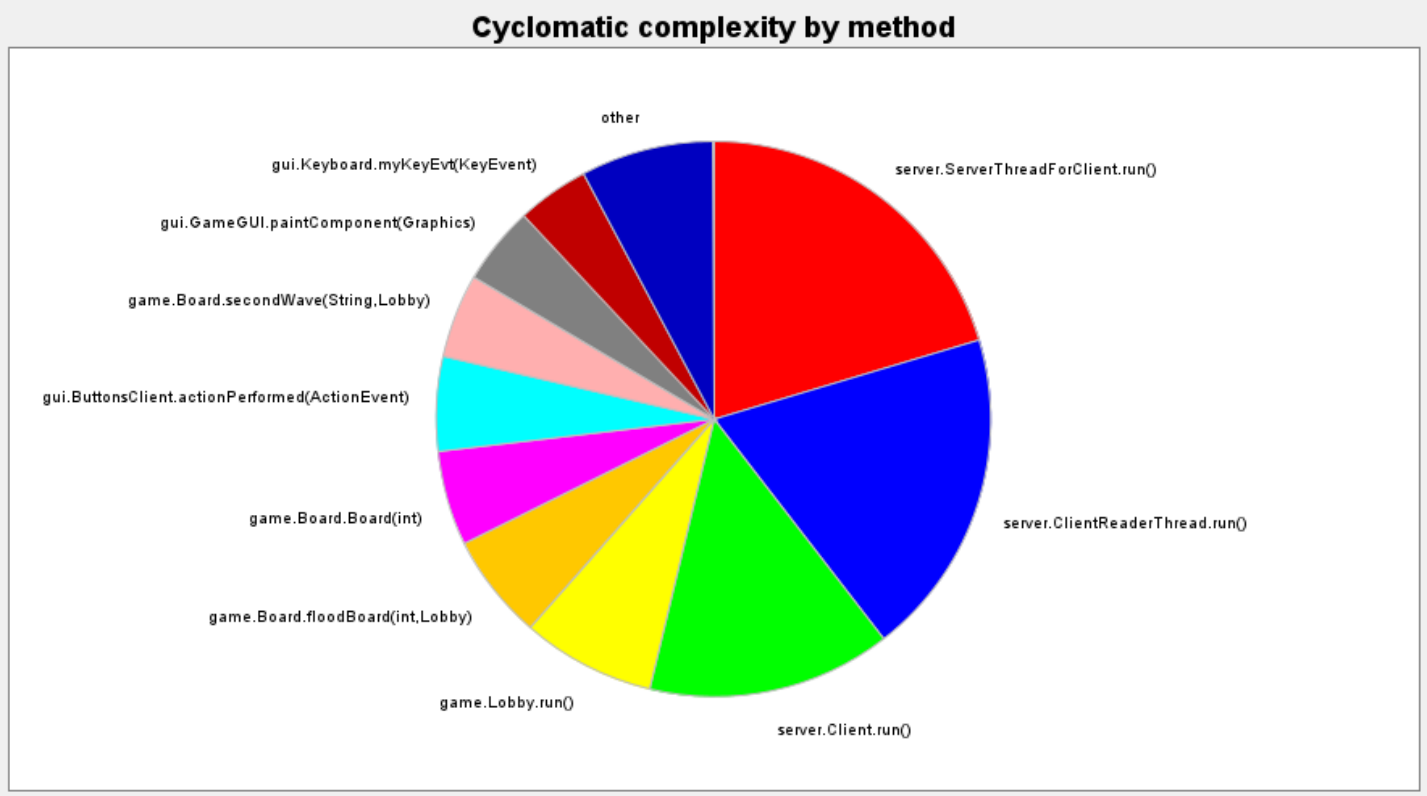
Die drei Metriken, für die wir uns entschieden haben sind Lines of Code pro Klasse, Code Coverage für Tests und Cyclomatic Complexity. Uns ist aufgefallen, dass unsere Java Klassen teilweise sehr viele switch-case-Verzweigungen beinhalten. Weshalb wir uns dazu entschieden haben, die zyklomatische Komplexität der Klassen bzw. Methoden zu messen. Für Meilenstein 3 wurde nur die zyklomatische Komplexität gemessen. Bis zum Meilenstein 4 wurden Lines of Code und Cyclomatic Complexity gemessen und zuletzt wurden bis zum Meilenstein 5 zusätzlich noch Code Coverage Messungen durchgeführt.

Code Coverage Messungen Meilenstein 4

Package	Class, %	Method, %	Line, %
game	57.1% (4/ 7)	25% (9/ 36)	17% (82/ 483)
server	36.4% (4/ 11)	34.5% (19/ 55)	15.4% (132/ 859)

Wie schon erwähnt, wurden bei Meilenstein 4 nicht alle relevanten Eigenschaften getestet, daher wurden nur 25 bis 35 % der Methoden der Packages Game und Server getestet.

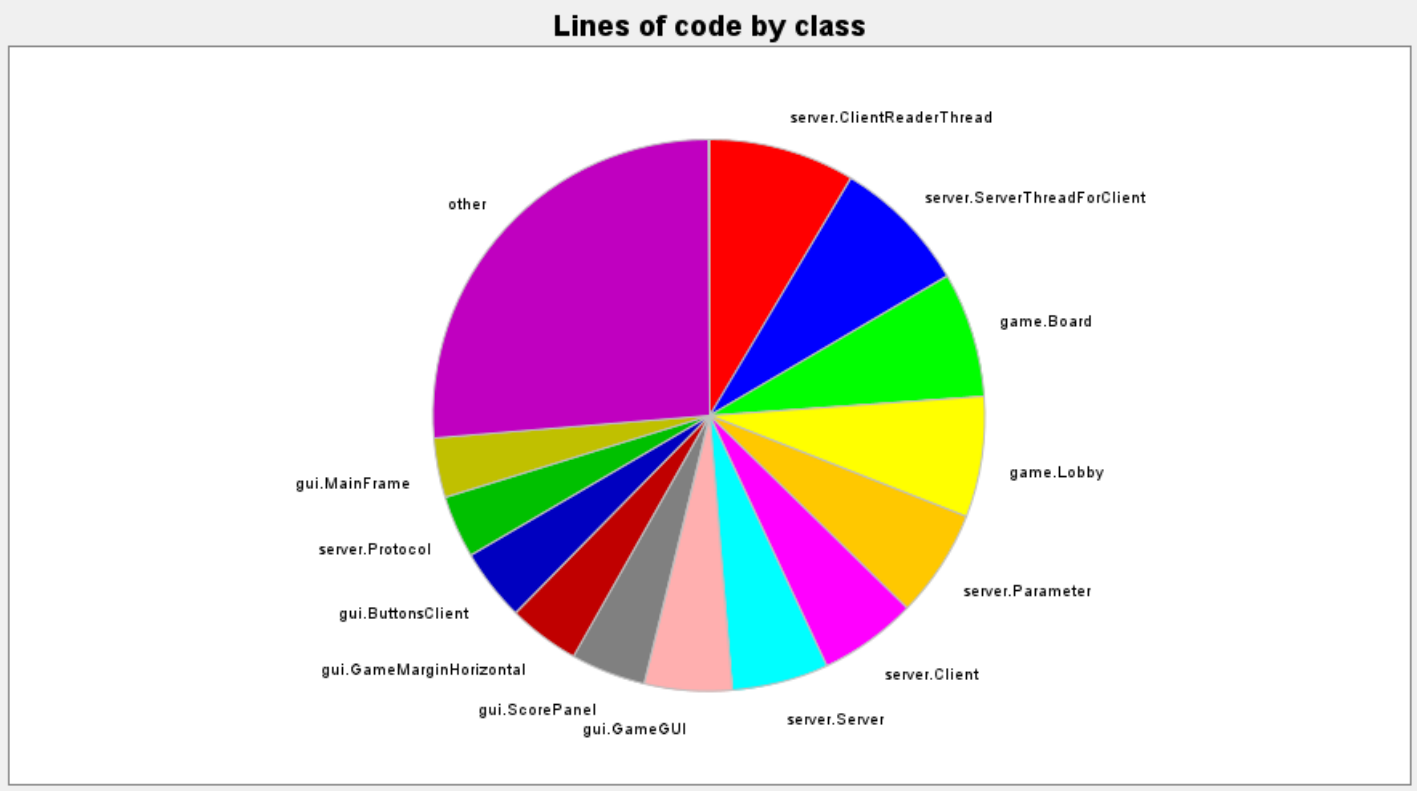
Meilenstein 4 zyklomatische Komplexität Messungen



Wie erwartet wiesen ServerThreadForClient und ClientReaderThread sehr hohe Werte auf. In unserem Projekt wird der In und Output bei Server und Client über die beiden Runnable Interfaces bearbeitet. Alle Runnable Interfaces weisen hohe Werte auf. Auf dem ersten Blick erscheinen die Klassen ServerThreadForClient, ClientReaderThread, Client und Lobby zwar recht gross und vielleicht auch unübersichtlich, doch dank sehr guter JavaDoc Dokumentation kann man sich auch als Aussenstehender sehr schnell einen Überblick verschaffen.

Wir hätten die Switch-case-Verzweigungen auch in einzelnen Methoden oder Klassen aufteilen und somit die zyklomatische Komplexität verringern können, doch wir haben uns als Gruppe darauf geeinigt, dass wir immer alle Switch-case-Verzweigungen in einer Klasse haben wollen, was für uns das Programmieren einfacher machte.

Meilenstein 4 Lines of Code Messung



Wie bereits bei der zyklomatischen Komplexität ersichtlich war, weisen die Klassen ClientReaderThread und ServerThreadForClient auch bei Lines of Code hohe Werte auf. Die Werte von der Klassen Server und Client verhalten sich wie, die Werte von ServerThreadForClient und ClientReaderThread mit dem Verhältnis 1 zu 1.

Code Coverage Meilenstein 5

Package	Class, %	Method, %	Line, %
game	57.1% (4/ 7)	44.4% (16/ 36)	32.6% (199/ 611)
server	45.5% (5/ 11)	42.9% (24/ 56)	20.7% (191/ 924)

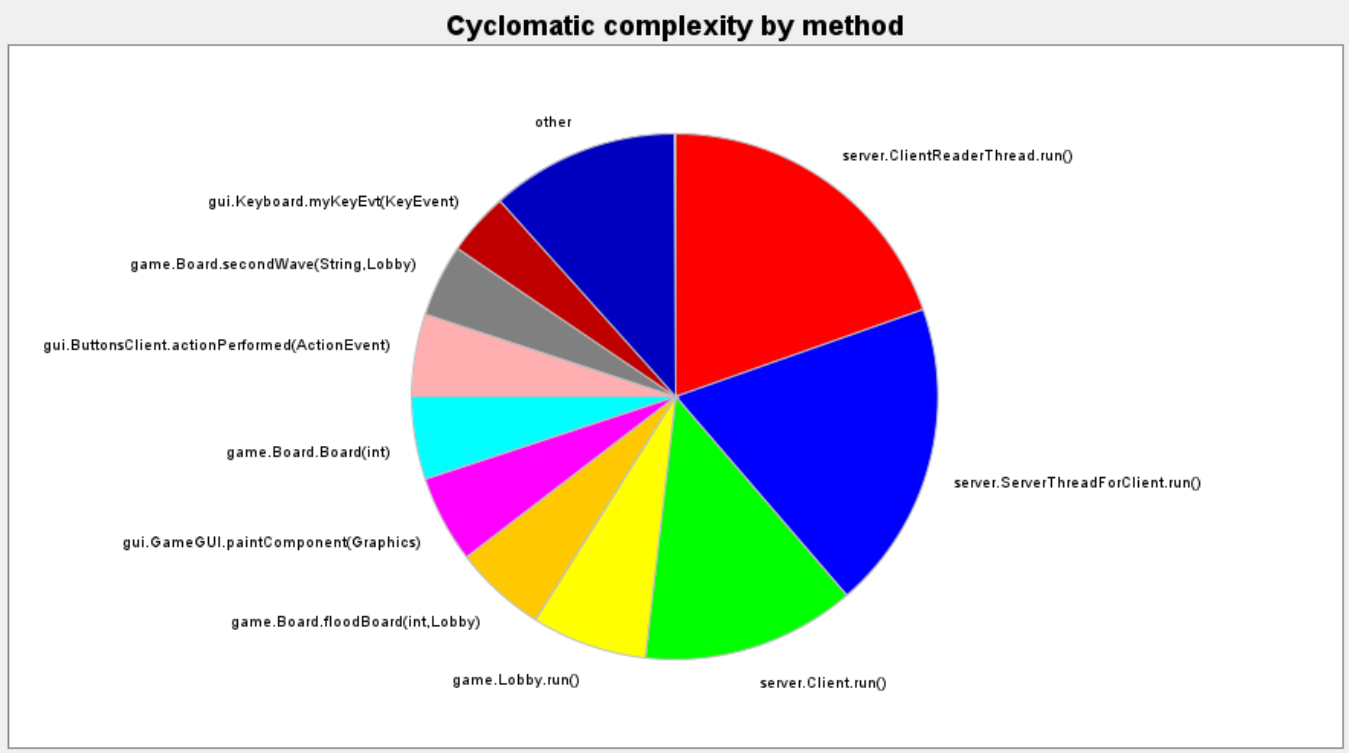
Class	Class, %	Method, %	Line, %
Board	100% (1/ 1)	70% (7/ 10)	51.3% (121/ 236)
Field	100% (1/ 1)	40% (2/ 5)	47.1% (24/ 51)
Lobby	100% (1/ 1)	54.5% (6/ 11)	18.9% (45/ 238)
PlayerTurtle	100% (1/ 1)	50% (1/ 2)	42.9% (9/ 21)

Class	Class, %	Method, %	Line, %
Parameter	100% (1/ 1)	45.5% (5/ 11)	27.3% (38/ 139)
Profil	100% (1/ 1)	40% (2/ 5)	64% (48/ 75)
Protocol	100% (1/ 1)	75% (3/ 4)	97.9% (46/ 47)
Server	100% (1/ 1)	66.7% (10/ 15)	40% (48/ 120)
ServerThreadForClient	50% (1/ 2)	44.4% (4/ 9)	5.9% (11/ 187)

Die Kernkomponenten unseres Spieles sind die Methode «moveTurtle» und die Events. In unserem Projekt gibt es drei Packages, nämlich Gui, Game und Server. Wir haben uns auf Game und Server fokussiert, wobei die Unit Tests 42 bis 44 % der Methoden beider Packages abdecken. Im Vergleich zu Meilenstein 5 wurden in Meilenstein 4

Bei der Klasse Lobby, welche für die Spielleitung zuständig ist, werden 55 % der Methoden dieser Klasse von Tests gedeckt. Bei der Klasse Board, welche für das Spielfeld und die Events verantwortlich ist, werden 70% der Methoden dieser Klasse getestet. Unser Ziel war es ungefähr die Hälfte der Methoden zu testen, was erfüllt wurde. Ausserdem wurden auch Tests für Sever-Client Kommunikation durchgeführt. Wobei die Klasse Parameter und Protocol getestet wurden.

Meilenstein 5 Messungen



Wie bei Meilenstein 4 haben auch hier die Klassen ClientReaderThread und ServerThreadForClient ähnliche Werte aufgewiesen.