

QA Report - Colliding Empires

Einführung

Um die Qualität von Colliding Empires sicherzustellen, haben wir von Anfang an Massnahmen getroffen, die uns helfen die Kohärenz des Projekts, bei wachsender Komplexität, sicherzustellen. Zu den Massnahmen, welche wir ergriffen haben, gehören zum Beispiel das Code Review und die Nutzung eines Logger. Im QA-Report wollen wir einen tieferen Blick auf die von uns erhobenen Metriken werfen und erläutern, was sie über die Qualität unseres Spiels aussagen. Die Erhebung dieser Metriken erfolgte jeweils für die einzelnen Meilensteine. Die Metriken, welche im Hauptteil des Reports näher kennenlernen werden sind:

- Anzahl Logging Statements normiert zu Zeilen Java Code
- Anzahl Methoden pro Klasse
- Anzahl Zeilen Code pro Methode
- Zyklomatische Komplexität von Methoden

Wie wir sehen werden, ist es nicht immer möglich sich an den gewählten Richtwert zu halten, und auch nicht immer sinnvoll. Die Metriken sollen uns aber helfen zu wissen wie komplex das Spiel ist, damit wir gegebenenfalls Anpassungen am Code und der Struktur durchführen können. Um die Metriken zu erheben, haben wir das IntelliJ-Plugin "Metrics reloaded" benutzt, welches eine grosse Anzahl von Metriken automatisch erheben kann.

Um die Coverage von unseren Unit-tests zu prüfen, haben wir ausserdem die Library Jacoco benutzt. Mit Jacoco können wir wissen wie viele Methoden und Pfade einer Methode mit unseren Tests geprüft werden. Ob die Tests aber sinnvoll sind, müssen wir selber bestimmen. Jacoco hilft aber sicherzustellen, dass wir keine Methoden vergessen.

Messungen und Auswertungen

(die Daten wurden zum Zeitpunkt der Abgabe des jeweiligen Meilensteins erhoben)

Logging Statements

Für den Logger haben wir die Library Log4j benutzt und er wurde primär zu Fehlersuche verwendet.

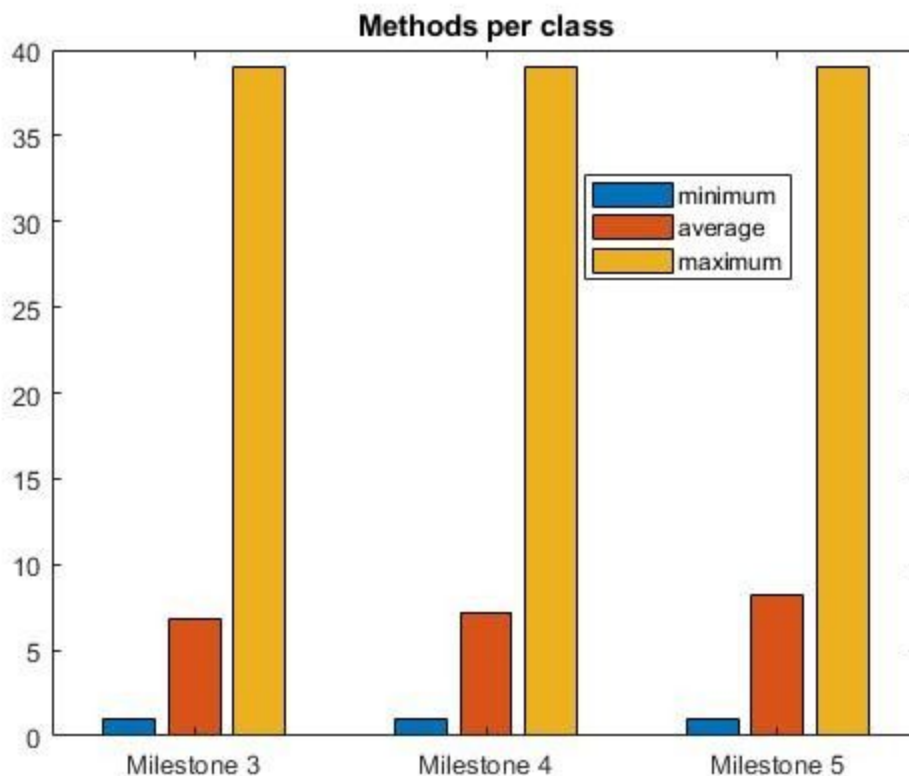
	Anzahl Logging Statements normalisiert zu Zeilen Java Code
Meilenstein 3	1 Statement zu 55 Zeilen Code
Meilenstein 4	1 Statement zu 196 Zeilen Code
Meilenstein 5	1 Statement zu 187 Zeilen Code

	Anzahl Logging Statements	Anzahl Zeilen Code
Meilenstein 3	34	1867
Meilenstein 4	37	7273
Meilenstein 4	44	8265

Es ist zu erkennen, dass die Grösse des Projekts viel schneller gewachsen als die Anzahl Logging Statements. Das ist damit zu begründen, dass wir den Logger fast nur benutzt haben, um den Grund eines entdeckten Bugs zu finden. Hier sehen wir Verbesserungspotential. Der Logger hätte konsequenter eingesetzt werden können, was uns möglicherweise Zeit hätte sparen können.

Methoden pro Klasse

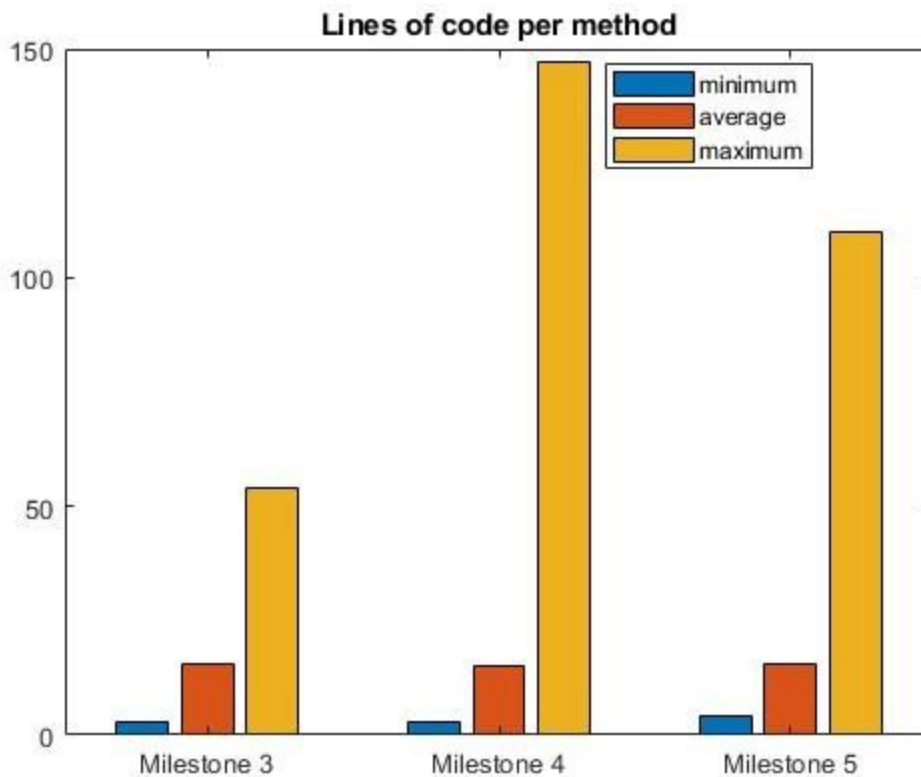
Die Anzahl der Methoden pro Klassen soll uns einen Überblick geben, wie viel eine Klasse macht. Eine Klasse mit sehr vielen Methoden wird unübersichtlich und sollte weiter aufgeteilt werden, zum Beispiel in eine Hauptklasse mit weiteren Unterklassen. Als Richtwert haben wir uns an der "Rule of 30" orientiert, welche besagt, dass eine Klasse nicht mehr als 30 Methoden haben soll.



Anhand des Diagramms ist zu sehen, dass wir den gewählten Richtwert überschritten haben. Der Grund liegt in unserer ClientMain Klasse. Diese Klasse verarbeitet die erhaltenen Protokollbefehle vom Server, was viele Methoden benötigt. Im Durchschnitt konnten wir diese Metrik auf circa 8 Methoden pro Klasse halten, womit wir zufrieden sind.

Anzahl Linen Code pro Methode

Die Metrik der Anzahl Zeilen Code pro Methode soll uns einen Überblick geben, wie viel eine Methode macht. Grosse Methoden werden schwieriger zu verstehen und die Gefahr ist gross, dass sich ein Fehler einschleichen kann. Zudem sind kleinere Methoden einfacher zu testen mittels Unit Tests. Auch bei dieser Metrik orientieren wir uns an der zuvor erwähnten "Rule of 30", also unter 30 Zeilen Code pro Methode.

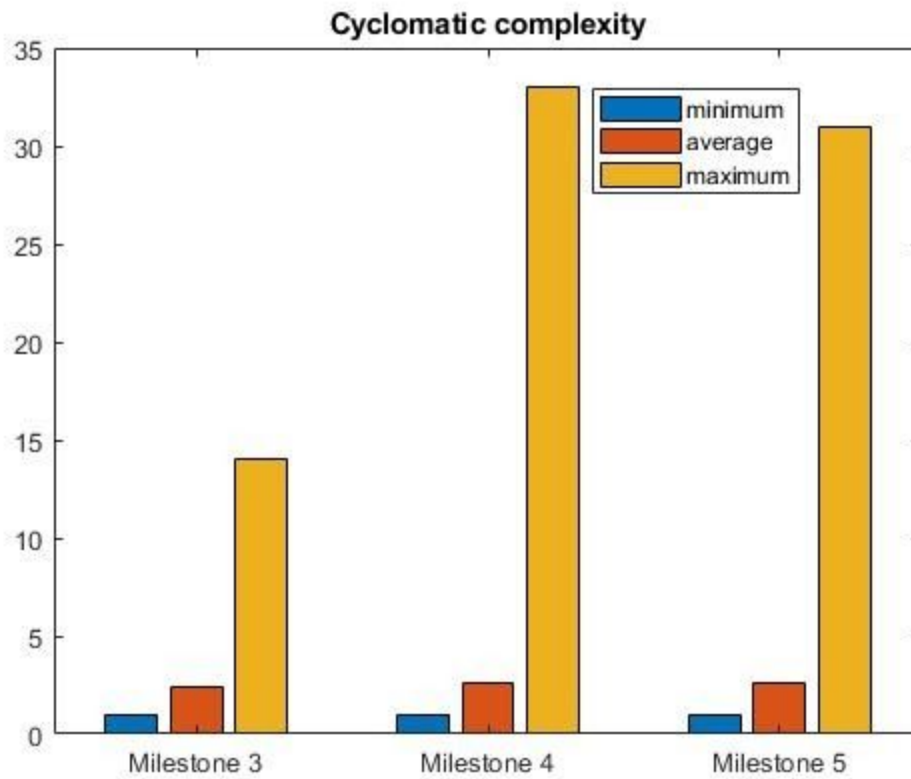


An den Daten ist zu erkennen, dass wir unser Ziel im Durchschnitt erreicht haben. Interessant zu sehen ist, dass wir auch bei dieser Metrik starke Ausreisser haben. Beim Meilenstein 4 ist dieser Ausreisser sehr extrem, welcher sich auf die execute Methode in der Executor Klasse bezieht. Bei der Bearbeitung von Meilenstein 5 haben wir versucht unseren Code ein bisschen zu vereinfachen. Dies hatte zur Folge, dass unsere grössten Klassen kürzer geworden sind.

Zyklomatische Komplexität

Als dritte Metrik, welche wir messen, haben wir die zyklomatische Komplexität der Methoden gewählt. Diese Metrik beschreibt, wie viele unabhängige Pfade eine Methode besitzt, um alle Statements abzudecken. Die Metrik beschreibt auch wie einfach oder schwer es ist eine

Methode zu verstehen. Wir haben uns entschieden für diese Metrik den Wert 10, falls möglich, nicht zu übersteigen.



Auch bei dieser Metrik erreichten wir im Durchschnitt unser Ziel. Bei dem Ausreisser dieser Metrik handelt es sich wie zuvor um die execute Methode in der Executor Klasse. Der Grund dieser hohen zyklomatischen Komplexität ist die switch Anweisung, welche die Protokollbefehle behandelt. Diesen Wert haben wir so erwartet und sind im Allgemeinen zufrieden mit dem Resultat.

Jacoco Code Coverage

Ein weiterer Teil unserer Qualitätssicherung ist die Implementierung von Unit Tests und die daraus resultierende Code Coverage. Dazu haben wir zu Meilenstein 3, 4 und 5 die Coverage der Unit Test gemessen.

Meilenstein 3

Colliding Empires

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
collidingempires_server		0%		0%	100	100	277	277	51	51	5	5
collidingempires_client_screens		0%		0%	88	88	276	276	51	51	7	7
collidingempires_client		0%		0%	87	87	268	268	43	43	1	1
collidingempires_client_net		56%		0%	39	42	85	122	5	8	1	2
collidingempires_client_util		0%		0%	36	36	120	120	27	27	8	8
collidingempires_server_net		68%		81%	13	30	40	88	8	14	1	3
collidingempires		0%		0%	10	10	28	28	3	3	1	1
Total	4'592 of 5'242	12%	320 of 346	7%	373	393	1'094	1'179	188	197	24	27

Zu diesem Zeitpunkt im Projekt existierte unser ingame Package der Game-Logik noch nicht. Da noch keine flächendeckenden Test erfordert waren, wurden nur vereinzelt Tests implementiert und die Coverage ist entsprechend klein.

Meilenstein 4

CollidingEmpires

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
collidingempires_server		1%		0%	223	229	703	725	74	80	6	8
collidingempires_client_screens		0%		0%	207	207	680	680	108	108	7	7
collidingempires_client		0%		0%	148	148	479	479	46	46	1	1
collidingempires_client_util		0%		0%	100	100	314	314	62	62	12	12
collidingempires_client_net		53%		0%	37	40	93	127	5	8	1	2
collidingempires_server_net		69%		81%	13	30	43	96	8	14	1	3
collidingempires		0%		0%	11	11	28	28	3	3	1	1
collidingempires_server.ingame		99%		92%	25	201	7	428	7	87	0	7
Total	10'662 of 13'320	19%	822 of 1'057	22%	764	966	2'347	2'877	313	408	29	41
















collidingempires.server.ingame

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Unit		87%		83%	4	10	3	23	3	6	0	1
Game		99%		93%	6	66	1	152	1	26	0	1
Building		93%		83%	3	8	2	20	2	5	0	1
Map		99%		92%	10	79	1	157	1	18	0	1
Field		100%		75%	2	17	0	37	0	13	0	1
Player		100%		100%	0	14	0	29	0	12	0	1
Container		100%	n/a	n/a	0	7	0	10	0	7	0	1
Total	18 of 1'926	99%	18 of 226	92%	25	201	7	428	7	87	0	7

Bei der Messung während Meilenstein 4 war die Game-Logik bereits implementiert und wir haben flächendeckend Tests für das ingame Package geschrieben. Mit dieser Coverage waren wir bereits sehr zufrieden.

Meilenstein 5

collidingempires.server.ingame

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Game		97%		89%	12	82	3	180	1	27	0	1
Map		99%		90%	16	96	3	188	1	19	0	1
Unit		87%		83%	4	10	3	23	3	6	0	1
Building		93%		83%	3	8	2	20	2	5	0	1
MapGenerator		100%		100%	0	29	0	42	0	5	0	1
Field		100%		90%	1	18	0	39	0	13	0	1
Player		100%		100%	0	14	0	29	0	12	0	1
Container		100%	n/a	n/a	0	7	0	10	0	7	0	1
Total	42 of 2'471	98%	30 of 338	91%	36	264	11	531	7	94	0	8

Aufgrund der Rückmeldungen der Unit Tests bei der Abnahme von Meilenstein 4 mussten wir die Tests anpassen. Wir hatten zu viel mit nur einem Test geprüft und mussten die Tests aufteilen, um kleinere Module zu testen. Die kleine Veränderung der Coverage ist darauf zurückzuführen, dass einige Test nicht sinnvoll waren und entfernt wurden. Andererseits wurden noch mehr Tests implementiert, um auch Randfälle zu prüfen.

Schlusswort

Im grossen Ganzen sind wir mit unserer Qualitätssicherung sehr zufrieden. Wir haben unsere Richtlinien für die Metriken wo möglich eingehalten, mussten diese jedoch überschreiten für unsere Executor und ClientMain Klasse, da wir keine einfachere Implementation nach unseren Richtwerten gefunden haben. Für alle anderen Klassen und vor allem bei der Game-Logik konnten wir jedoch sicherstellen, dass die Richtlinien eingehalten werden. Die Metriken aus den Metrics reloaded Plug-in sind somit von Meilenstein 3 bis Meilenstein 5 mehr oder weniger konstant geblieben. Das sieht man vor allem, wenn man den Durchschnitt betrachtet, welcher sich nicht wesentlich verändert hat.

Bei den Unit-Tests hatten wir mehr Probleme, da uns die Jacoco Code Coverage keine Informationen über die Qualität der Tests gibt. Darum ist die Code Coverage zwischen Meilenstein 4 und Meilenstein 5 fast gleich geblieben, die Qualität der Tests ist aber Meilenstein 5 bedeutend höher. Das liegt am Feedback unserer Tutoren, welche uns erklärten, wie man Unit-Tests richtig schreibt.

Wenn wir etwas an der Qualitätssicherung unseres Projekts verbessern könnten, wäre das wahrscheinlich unseren Umgang mit dem Logger. Diesen haben wir im Projekt nur spärlich benutzt, um punktuell Bugs zu finden. Das war sehr nützlich für uns und wir konnten damit auch Bugs beheben, von denen wir sonst die Ursache nicht gefunden hätten. Wir könnten uns jedoch vorstellen, dass wenn wir den Logger flächendeckend eingesetzt hätten, es gar nicht zu diesen Bugs gekommen wäre.

Wir haben aus dem Projekt gelernt wie wir sicherstellen können, dass ein Programm unseren Vorgaben entspricht und sind uns sicher, dass wir bei einem nächsten Projekt dieses Wissen einsetzen können und dabei Zeit sparen können.