Quality Assurance Konzept – Colliding Empires

Einleitung

Um die Qualität von Colliding Empires sicherzustellen haben wir uns für verschiedene Massnahmen und Tools entschieden, auf Basis der Vorgaben des Projekts. Hier finden sie die benutzten Plugins und erste Messungen, sowie deren Visualisierungen.

Code Review

Um eine hohe Qualität unseres Codes sicherzustellen werden wir Code Reviews durchführen. Dieses Gegenlesen hat das Ziel, dass der Code einheitlich aufgebaut ist und somit leichter erweitert werden kann. Wenn der Code mehr oder weniger gleich aufgebaut ist, ist auch das Schreiben der Unit Test vereinfacht. Das Code Review wollen wir mindestens für die fundamentalen Codestücke durchführen. Dafür haben wir eine Checkliste erstellt:

- Der Code ist lesbar und verständlich
- Der Code ist ausreichend dokumentiert
- Der Code ist leicht ausbaubar
- Gleiche Funktionalität ist noch nicht vorhanden

Logger

Als Logger benutzen wir die Log4j Libarary. Der Logger ist so konfiguriert, dass in einen Log File alle Statements mit dem Level Error oder höher angezeigt werden und in einem weiteren File die Statements mit dem Level Trace oder höher.

Im Allgemeinen soll uns der Logger helfen Fehler schneller zu finden. Genauer soll derTrace Log dazu dienen den Programmverlauf folgen zu können und der Error Log soll helfen schwerwiegende Fehler zu finden.

Metrics Reloaded

Mittels dem Plugin Metrics Reloaded in Intellij IDEA werden wir drei verschiedene Metriken regelmässig erstellen und auswerten. Dabei versuchen wir folgende Daumenregeln einzuhalten.

- Eine Metrik, die wir messen ist die zyklomatische Komplexität. Sie zeigt uns wie viele unabhängige Pfade eine Methode besitzt. Wenn möglich wollen wir die Zahl dieser Pfade unter Zehn halten. Je tiefer diese Zahl ist, desto weniger Tests benötigt man, um alle Pfade zu testen.

- Die zweite, zu messende Metrik ist die Anzahl Linien Code pro Methode. Dies soll dazu dienen, dass wir nicht zu lange Methoden schreiben, die unübersichtlich werden. Es ist also bevorzugt zwei kleinere und verständlichere Methoden zu schreiben als eine Lange. Die Daumenregel, welcher wir folgen wird bei der nächsten Metrik vorgestellt.

- Die dritte Messung, die wir durchführen ist die Anzahl Methoden pro Klasse. Einzelne Klassen sollen nicht zu viel Funktionalität haben. Zu viele Methoden in einer Klasse macht es schwieriger diese zu verstehen. Falls die Anzahl Methoden zu gross ist sollte man sich überlegen die Klasse aufzuteilen. Zum Beispiel in eine Hauptklasse und eine Unterklasse. Die Daumenregel, welche wir versuchen zu befolgen heisst «Rule of 30», vorgestellt von Jim Bird. Diese Regel besagt, dass Methoden durchschnittlich nicht mehr als 30 Zeilen Code beinhalten sollten und Klassen nicht mehr als 30 Methoden haben sollten. Es hat das Ziel die

Komplexität einzelner Elemente in Grenzen zu halten und dient somit auch der Fehlerprävention.

Diese Regeln sind nicht absolut. Es darf also Ausnahmen geben, wenn es schlicht nicht möglich oder sinnvoll ist sie einzuhalten.

Unit Test

Es muss sichergestellt sein, dass die essenziellen Programmteile von Colliding Empires korrekt Funktionieren. Um dies zu testen werden Unit Test verwendet. Zu den wichtigsten Teilen des Spiels gehören unter anderem:

- die Spiellogik
- Encoding, Decoding, Validating
- Lobby/Player Management

Das Schreiben von Unit Tests ist nicht immer einfach. Es hängt auch vom zu testendem Code ab. Eine einfache Lösung wäre Tests schon vor dem eigentlichen Implementieren zu schreiben. Um uns aber nicht zu stark auf die Tests zu konzentrieren, sondern auf das Programm selbst, das wir schreiben, nehmen wir uns vor die Unit Tests zumindest im Hinterkopf zu behalten.

Jacoco

Jacoco ist eine Library um zu ermitteln, welchen Anteil vom Programm durch Unit Tests geprüft wurde. Natürlich wäre es am besten man würde den gesamten Code möglichst flächendeckend zu testen. Das ist aber nicht realistisch und auch nicht nötig in unserem Fall. Da das GUI nur schwer mit Unit Tests prüfbar ist werden wir es manuell testen müssen. Im Fokus stehen die Programmteile, welche am meisten benutzt werden. In diesen Teilen wird die Code Coverage von 100% angestrebt. Da erst ab Meilenstein 4 flächendeckende Tests vorausgesetzt sind, wird die Messung der Coverage bei Meilenstein 3 vermutlich noch nicht dem angestrebten Ziel entsprechen.

Erste Messungen

Stand 30.3.2020

	Lines of Code pro Methode	Methoden pro Klasse	Zyklomatische Komplexität (Methoden)
Minimum	6 (ServerInputThread.terminate)	12 (client.screens.Screencontroller)	1 (server.Executer.terminate)
Durchschnitt	15.53	21.35	2.42
Maximum	54 (Main.main)	73 (server.Executer)	9 (Main.main)

	Logging Statements normalisiert zu Lines of Code (1867 Lines)
Error Level (12 Statements)	1 Statement für 155 Linien Code
Trace Level (22 Statements)	1 Statement für 85 Linien Code
Total (34 Statements)	1 Statement für 55 Linien Code

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