Konzept zur Qualitätssicherung

Buddler Joe

April 10, 2019

Inhalt

1 Einleitung

Wie in der Vorlesung vom 22.03.2019 vorgezeigt, kann Fehlerhafte Software schnell zu grossen Schäden und Komplikationen führen. Natürlich wird sich der Schaden bei unserem Projekt in etwas kleinerem Rahmen als die Millionenschäden bewegen, jedoch ist es auch für uns wichtig, eine starke Qualitätssicherung zu haben.

Dafür werden wir uns selber hohe Qualitätsanforderungen stellen, welche wir mit starken Methoden überprüfen und einhalten werden. Dafür haben wir ein umfangreiches Qualitätssicherungskonzept erarbeitet.

In den folgenden Seiten werden wir uns mit unseren Massnahmen zur Qualitätssicherung befassen. Dazu gehören unsere Code Standards, unseren Issue Trackern, unserer Kommunikationsstrategie, unserem Umgang mit GIT, den Unit Tests und abschliessend das Javadoc.

Wir werden uns in diesem Dokument primär mit unseren Umsetzungsstrategien befassen und unseren Plänen, wie wir die Qualität unseres Spieleprojekts auf einem hohen Niveau halten.

1.1 Zielsetzung

Wir haben uns zur Zielsetzung genommen, so viel wie möglich aus diesem Softwareprojekt zu lernen. Dies heisst vor allem, dass wir regelmässig alle unsere Strategien überprüfen werden und diese bei Bedarf anpassen werden.

Weiterhin werden wir uns achten, soviele effektive Massnahmen wie möglich einzusetzen, damit wir als Gruppe den grössten Lerneffekt haben. Dadurch werden wir mit einem effektiven und geplanten Ablauf den Qualitätsstandard hoch halten.

Zusätzlich werden wir in der Kommunikation auf klare Verantwortlichkeiten setzen. Dadurch ist allen ihre Rolle klar definiert und jeder kann seine Aufgaben pflichtbewusst umsetzen.

Dies alles sollte unserem Ziel eines möglichst fehlerfreien Programmes helfen und auch helfen, reflektierend und überlegt zu arbeiten.

2 Massnahme: Code Standards

2.1 Eingliederung der Massnahme

Coding Standards sind Teil des konstruktiven Qualitätsmanagements und dort den organisatorischen Massnahmen angehängt. Coding Standards setzten Richtlinien und Standards für den Umgang mit Quellcode und Entwicklungsumgebungen fest. Coding Standards haben viele Vorteile, welche direkt zur Qualität einer Software beitragen. Die Vorteile lassen sich grob in ästhetische und nicht-ästhetische Kategorieren unterteilen.

Ästhetische Vorteile ergeben sich durch die vereinfachte und konsistente Lesbarkeit des Quellcodes über das ganze Projekt hinweg. Dies macht überprüfen von nicht selbst geschriebenem Code (oder vor einer Weile geschriebenen Codes) im Projekt einfacher und effizienter. Eine gute Struktur erleichtert ausserdem die Navigation im Quellcode.

Nicht-Ästhetische Vorteile sind etwa, dass sich weniger Fehler im Code einschleichen können, wenn man sich an Standards hält (z.B. immer Klammern setzten, auch wenn das If-Statement nur eine Zeile ist).

2.2 Wahl des Standards

Aufgrund grosser Beliebtheit und auf anraten unseres Tutors werden wir den **Google Java Style Guide** für unser Projekt verwenden:

`https://google.github.io/styleguide/javaguide.html`

Das Dokument ist kurz, leicht verständlich und praxis-erprobt.

2.3 Ergänzende Standards

Zusätzlich zu den Standards im Google Java Style Guide machen wir uns Gedanken zu Standards welche für unser konkretes Projekt und unseren Lernerfolg sinnvoll sind. Diese ergänzenden Standards sind bisher an Sitzungen und im Discord besprochen worden, sollen aber in Zukunft zentral nieder geschrieben werden. Eine nicht abschliessende Liste mit Standards und Konzepten, welche wir bisher besprochen haben und an welche wir uns zu halten versuchen.

2.3.1 Information Hiding

Wir legen sehr viel Wert auf das Prinzip des Information Hiding. Jede Klassenvariable ist entweder privat oder es liegt ausführliche Dokumentation mit guten Gründen vor falls die Variable nicht privat ist.

2.3.2 APIs und Schnittstellen

Wird ein Package mit mehren Klassen erstellt, sollte die API oder Schnittstelle dafür - falls benötigt - sich auf so wenig Klassen wie möglich beschränken. Wenn immer möglich sollte die API in einer "Master-Klasse" zusammengefasst sein.

2.3.3 Zuständigkeiten

Jede Klasse hat eine eigene Zuständigkeit und Rolle, welche sich nicht mit einer anderen Klasse überschneiden sollte. Die Logik einer Methode sollte immer in der Klasse geschrieben werden, bei welcher die Zuständigkeit am besten gegeben ist. Beispiel: Die methode sendToLobby() aus der abstrakten Packet Klasse fällt in den Zuständigkeitsbereich der ServerLogic. Folgich wird die Methode dort implementiert und aus der Packet Klasse nur aufgerufen.

2.4 Konkrete Umsetzbarkeit im Projekt

2.4.1 Ziel für Meilenstein 3

Alle Teammitglieder arbeiten mit IntelliJ, welches Code Standards erzwingen kann bzw. es kann Verletzungen des Standards als Warnungen anzeigen. Bis zum 3. Meilenstein wollen wir folgende Ziele erreicht haben:

- Alle Mitglieder haben das den Google Java Style Guide gelesen und verstanden
- Alle Mitglieder haben das *checkstyle* Plugin für IntelliJ installiert
- Sämtlicher Code ist so umgeschrieben, dass er sich an den Code Standard hält
- Neuer Code wird direkt im richtigen Stil geschrieben

- Wir haben die ergänzenden Standards ausformuliert und in einem eigenen Dokument niedergeschrieben
- An mindestens einer Sitzung werden Code Standards besprochen und potentiell erweitert

2.4.2 Ziel für Meilenstein 4

Die für den dritten Meilenstein gesetzten Ziele sind erreicht. Das *checkstyle* wurde regelmässig von allen Teammitgliedern eingesetzt. Wie man der Graphik entnehmen kann, bewegt sich die Fehlerzahl fast im Nullbereich. Vor dem Anwenden des *checkstyle* plugins befanden sich knapp 9'000 Fehler im Quellcode, welche korrigiert wurden. Für den vierten wie auch den fünften Meilenstein möchten wir unseren Quellcode möglichst fehlerfrei beibehalten und natürlich vor jedem Merge vom lokalen Branch in den Master einen fehlerfreien Code bereitstellen.

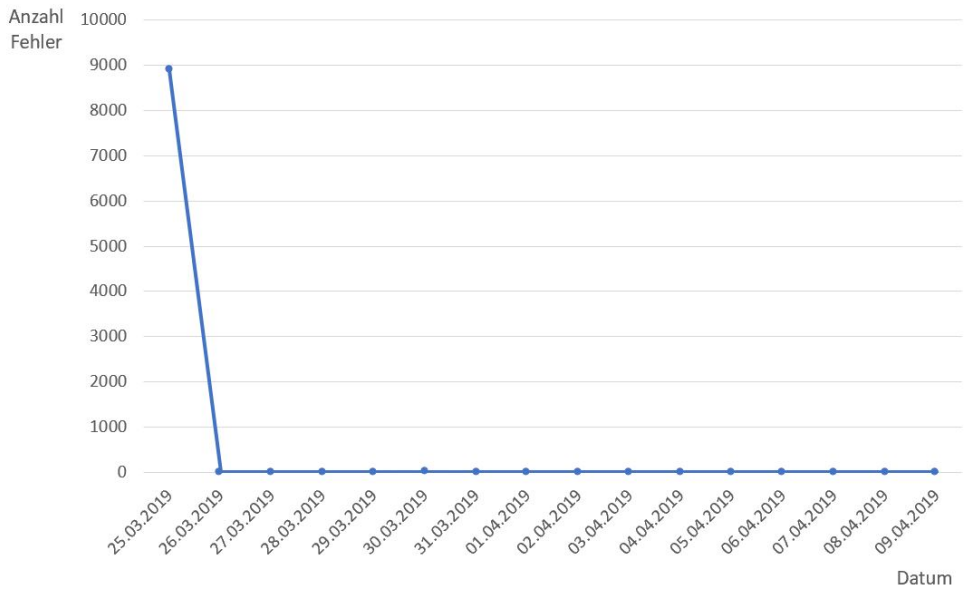

Figure 1: Anzahl Fehler im Quellcode

2.5 Kontrolle der Massnahme

2.5.1 Kontrolle des Code Styles

Die Kontrolle hier ist simpel: Je weniger Warnungen das Checkstyle Plugin anzeigt, desto besser ist die Massnahme umgesetzt; mit dem Ziel den kompletten Code ohne undokumentierte Style Warnungen zu haben.

2.5.2 Kontrolle der erweiterten Standards

Wir haben keine automatisierten Methoden um eine quantitative Kontrolle durchzuführen. Stattdessen vertrauen wir auf gegenseitige Kontrolle im Team. Wir machen uns gegenseitig auf potentielle Verletzungen unserer Standards aufmerksam und suchen bei Bedarf gemeinsam eine Lösung. Im Discord ist die Diskussion für alle, auch nachträglich, ersichtlich und sie dient gleichzeitig als ein Archiv für ähnliche Situationen in der Zukunft.

3 Massnahme: Issue Tracker

3.1 Eingliederung der Massnahme

Issue Tracking und Bug Reporting sind Teile des analytischen Qualitätsmanagements und dort den analysierenden Verfahren angehängt. Sie dienen dazu, im Programm entdeckte Fehler für alle Entwickler übersichtlich und standardisiert abzulegen. Der Issue Tracker erlaubt ebenfalls das gruppieren ("taggen") der Reports und ein Zuweisen von Team Mitgliedern an die Fehler.

3.2 Formattierung eines Issues

Folgende Informationen sollten in jedem Issue vorhanden sein:

Branch und Version. Falls nicht auf dem master, auf welchem Branch und auf welcher Version des Branchs das Problem entdeckt wurde.

Beschreibung Wie genau äussert sich das Problem? Welche Teile des Programms sind betroffen?

Logs Logs und Konsolenoutput vor, während und nach dem Problem, wenn vorhanden.

Reproduktion Welche Schritte sind nötig, um das Problem herbeizuführen und wie zuverlässig ist die Reproduktion?

Erwartung Was ist das erwartete Verhalten?

3.3 Konkrete Umsetzbarkeit im Projekt

3.3.1 Ziel für Meilenstein 3

Wir haben oft relativ klar aufgeteilte Zuständigkeiten, wer welchen Code schreibt. Issues können so in unserem Projekt direkt an die verantwortliche Person zugeteilt werden. Bis zum 3. Meilenstein wollen wir folgende Ziele erreicht haben:

- Sinnvolle Labels für Issues sind erstellt
- Jedes Mitglied hat mindestens einen Issue nach Vorlage erstellt
- Der Issue Tracker ist mit unserem Discord verbunden

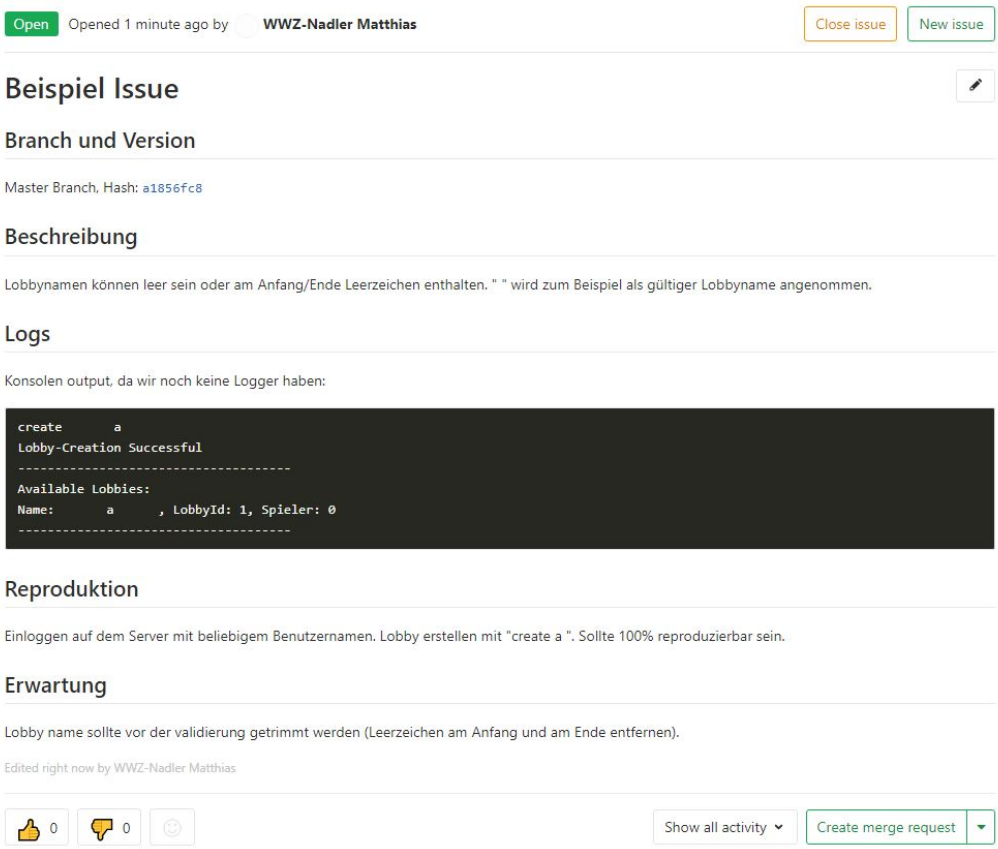

Figure 2: Beispiel eines vollständigen Issue Reports

3.3.2 Ziel für Meilenstein 4

Für den dritten Meilenstein hat jeder schon mindestens ein Issue nach Vorlage erstellt und auch mit sinnvollen Labels vermerkt. Für den vierten Meilenstein möchten wir den Issue Tracker aktiver benutzen.

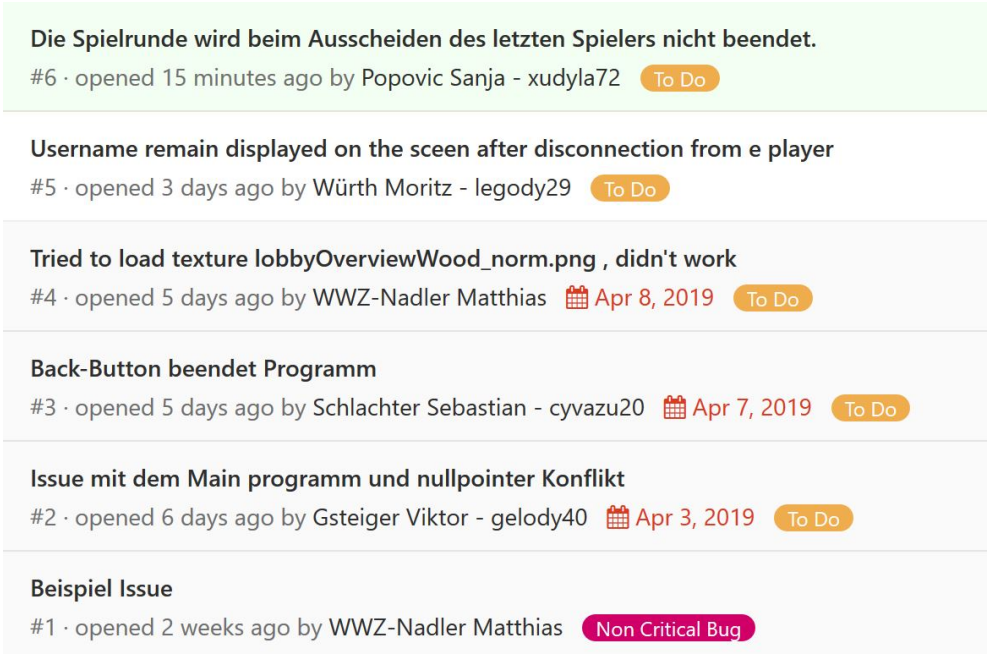

Figure 3: Issuereports

3.4 Kontrolle der Massnahme

Wird ein Issue nicht ordnungsgemäss formatiert, kann dies direkt dem Mitglied mitgeteilt werden. Jedem Issue kann eine Person und ein Datum zugeteilt werden, was die Chance, dass ein Issue vergessen geht, sehr stark reduziert.

4 Massnahme: Kommunikation

4.1 Eingliederung der Massnahme

Die Kommunikation ist in solch einem Projekt ein wichtiger Teil der konstruktiven Qualitätsmanagement. Da die Gruppe aus fünf Personen besteht, ist die Kommunikation untereinander ein wichtiger Bestandteil. Durch eine gründliche Kommunikation kann viel Zeit eingespart werden. Durch fehlende Absprache mit anderen Gruppenmitgliedern können Fehler generiert werden, welche viel Zeit kosten, um sie wieder zu beheben.

4.2 Wahl des Standards Kommunikationsmittel

Unsere Standardkommunikationsmittel sind Discord und Whatsapp. Discord wird von der Gruppe verwendet um eine klare Kommunikationsstruktur einzuhalten, bei welcher verschiedene Kanäle erstellt werden für unterschiedliche Themen. Dadurch ist immer klar abgegrenzt, was in welchem Chat Verlauf besprochen wird. Eine schnelle Kommunikation ist gewährleistet dadurch das Discord sowohl von Telefon als auch vom Computer verwendet werden kann. Somit ist eine rascher kommunikationsaustausch gewährleistet. Zudem biete Discord auch einen Voice Chat Funktion an, welche wir nützen und mündlich zu kommunizieren, während wir am Programmieren sind. Aufkommende Fragen oder Probleme werden so schneller beantwortet beziehungsweise gelöst. Whatsapp hingegen wird grundsätzlich für allgemeine absprechen benützt um zum Beispiel eine Besprechung zu planen. Die Kommunikationanteil via Whatsapp hält sich aber eher gering.

4.3 Ergänzende Kommunikation

Natürlich sollte ein Projekt nicht nur über Textnachrichten und Sprachchchat verlaufen. Deshalb sind gemeinsame Treffen sehr wichtig um einen Überblick über das Projekt zu behalten. So können weitere Schritte koordiniert geplanen werden. Kommende Aufgaben werden besprochen und auf die Gruppenmitglieder verteilen. Zudem treffen wir uns jede zweite Woche um gemeinsam weiter an unserem Projekt weiterzuschreiben. Durch die direkte Anwesenheit der anderen Gruppenmitgliedern können bereits geschriebene Klassen und Methoden gemeinsam überarbeitet werden. Dazu kommt die Kontrolle der anderen Mitgliedern welche Fehler finden, welche man alleine vielle-

icht übersehen hätte. Solch eine Kontrolle ist wichtig in einem Projekt. Ohne solch eine Kontrolle können Fehler passieren, welche in einem späterem Zeitraum des Projekt viel Zeit und Arbeit kosten um sie zu beheben.

4.4 Konkrete Umsetzbarkeit im Projekt

In unserem Projekt legen wir viel Wert darauf eine gute und stabile Kommunikation zu führen. Uns gegenseitig zu helfen und auch gegenseitig zu verbessern. Unsere Gruppe ist deshalb täglich in Discord am kommunizieren und treffen uns mindestens einmal in der Woche um den Stand der Dinge zu prüfen. Diese Wöchentlichen Treffen sind uns sehr wichtig und dürfen im verlauf des Projekts nicht vernachlässigt werden. Das gemeinsame arbeiten am Projekt machen wir jede zweite Woche.

4.5 Kontrolle der Massnahme

Eine Kontrolle fällt wesentlich einfach aus. Wir halten uns an unsere Vereinbarung uns jede Woche zu treffen und den Kontakt auf Discord weiterhin aufrecht zu halten.

5 Massnahme: GIT

5.1 Eingliederung der Massnahme

GIT ermöglicht das gleichzeitige, koordinierte Arbeiten aller Teammitglieder an der Software. Man kann effizienter arbeiten und ist bis zu einem gewissen Punkt unabhängig von den anderen Teammitgliedern, weil jeder auf seinem eigenen Branch arbeiten kann. Trotzdem bleibt die Interaktion erhalten, denn es wird allen Mitgliedern ermöglicht, Einblick in die Arbeit seines Teamkollegen zu erhalten. Erachtet man seinen Branch als einsatzbereit, so kann man ein Mergerequest erstellen und den Quellcode gegenlesen lassen. Ordnung und Struktur werden gewährleistet, indem man den Quellcode im Master nicht bearbeiten darf.

5.2 Konkrete Umsetzbarkeit im Projekt

Jedes Teammitglied benutzt die graphische Oberfläche GitKraken, welches das Ganze übersichtlicher gestaltet. Die Commits jedes Teammitglieds werden verfolgt und auf dem Bildschirm visualisiert. Durch das Anklicken der jeweiligen Commits, werden die Quellcodeabschnitte visualisiert und neue Bereiche Grün, gelöschte Bereiche mit Rot markiert. Wichtig ist es, jeden Commit mit einer ausschlaggebenden Nachricht zu versehen.

5.3 Kontrolle der Massnahmen

Die Visualisierung von GitKraken stellt uns eine detaillierte Dokumentation über die Fortschritte des Projekts wie auch über den Einsatz jedes Teammitglieds zur Verfügung. Diese sind für alle Projektteilnehmer einsehbar.

6 Massnahme: Unit Tests

6.1 Eingliederung der Massnahme

Unittests sind Teil des analytischen Qualitätsmanagements. Durch Unittests können einzelne Klassen und Module auf Fehler getestet werden. Durch solche Tests versucht man Fehler von Grund auf zu finden, indem man eine Klasse als kleinste Einheit testet. Auf diese Art und Weise kann man prophylaktisch bei Interaktion aller Klassen grössere Fehler in der Software beheben und somit auch ein ewiges Suchen des Entstehungsorts des Fehlers im Quellcode bis zu einem gewissen Punkt vermeiden.

6.2 Konkrete Umsetzbarkeit im Projekt

6.2.1 Ziel für Meilenstein 4

- Es sollen alle Teammitglieder mit JUnit und einer weiteren Bibliothek arbeiten.
- Die Autoren der Unit Tests müssen die Funktionen der zugeteilten Klassen detailliert kennen und verstehen.
- In einer Datei werden die Zuständigkeiten für die Unit Tests niedergeschrieben, um Überschneidungen zu vermeiden und Lücken zu verhindern.
- An der ersten Sitzung für den vierten Meilenstein werden die Zuständigkeiten besprochen.
- Bis zum nächsten Meilenstein sollen 40% des Codes mit Tests abgedeckt sein.

6.3 Kontrolle der Massnahmen

Um Unübersichtlichkeit zu vermeiden, wird in einem eigenen Ordner eine Datei angelegt, in welcher die Zuständigkeiten aller Teammitglieder bezüglich der Unit Tests aufgelistet sind. Dadurch kann man sicherstellen, dass auch alle relevanten Klassen getestet werden. In einer weiteren Datei soll notiert werden, welche Unit Tests schon geschrieben wurden. Es soll dabei angegeben werden,

- Wer der Autor ist.

- Was diese Anwendung genau testet.
- Wann der Test erstellt wurde.
- Ob der Test schon ausgeführt wurde.
- Welche Bugs bei der Ausführung Nr. * aufgetreten sind. (Ausdruck)
- Was im Quellcode korrigiert wurde. (GIT)
- Alle Reports müssen sich als Ausdruck im Ordner befinden

7 Massnahme: Javadoc

7.1 Eingliederung der Massnahme

Javadoc ist ein Teil des konstruktiven Qualitätsmanagements. Jedem Teammitglied wird es durch das Dokumentieren des Quellcodes ermöglicht, sich schnell in den Quellcode einzulesen und mit wenig Zeitaufwand die Funktion der jeweiligen Klasse oder der jeweiligen Methode zu erfahren. Auf diese Art und Weise wird auch Ordnung im Quellcode geschaffen.

7.2 Konkrete Umsetzbarkeit im Projekt

7.2.1 Ziel für Meilenstein 3

Der Autor der jeweiligen Methode oder der jeweiligen Klasse ist für deren Dokumentation zuständig, da dieser selbst am besten die Funktion kennt. Wird eine Klasse als «abgeschlossen» erachtet, so wird die HTML-Datei aktualisiert.

7.2.2 Ziel für Meilenstein 4

Bisher wurde der Stand des Javadocs nicht sofort von allen Teammitgliedern aktualisiert, das heisst, dass der Quellcode nicht immer auf dem neusten Stand ist. Die Aktualisierung erfolgte erst nach ein paar Tagen. Bis zum nächsten Meilenstein möchten wir Disziplin schaffen und spätestens nach jedem Merge in den Master ein aktuelles Javadoc haben.

7.3 Kontrolle der Massnahmen

Das Javadoc ist jederzeit in einem Ordner für alle Gruppenmitglieder einsehbar.

8 Massnahme: Logging

8.1 Eingliederung der Massnahme

Logging ist eine Massnahme zur erleichterten Fehlerbehebung. Durch verschiedene Log Levels ermöglich ein effektives Logging Tool eine schnelle Fehlersuche.

Durch eine Echtzeitverfolgung lassen sich ausserdem bei einem aktiven Review des codes schnell Fehler entdecken. Zusätzlich gibt es Tools zum Filtern, Suchen, Hervorheben und viele mehr, welche die Fehlersuche noch einfacher machen.

8.2 Wahl des Standards

Aufgrund grosser Beliebtheit und auf Hinweis in der Vorlesung vom 22.3.2019 werden wir das Tool **slf4j** für unser Projekt verwenden:

`https://www.slf4j.org/`

Das Tool ist einfach einsetzbar, Open Source, modular und einfach Bedienbar.

8.3 Konkrete Umsetzbarkeit im Projekt

8.3.1 Ziel für Meilenstein 3

Für regelmässige Code Reviews soll das Tool eingesetzt werden, um isolierte Teile des Codes zu überprüfen. Dafür sollte vor den jeweiligen Reviews das Logging eingerichtet werden.

8.3.2 Ziel für Meilenstein 4

Der Logger (slf4j) ist in unserem Projekt eingebaut und schon benutzt worden. In einem Ordner namens log sind die Textdateien zu finden, welche dort für 30 Tage lokal gespeichert bleiben.

```
22:50:06.073 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 1
22:50:07.075 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 2
22:50:08.076 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 3
22:50:09.076 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 4
22:50:10.078 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 5
22:50:11.079 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 6
22:50:12.079 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 7
22:50:13.081 [Thread-0] DEBUG net.playerhandling.PingManager - Number of unanswered pings: 8
22:50:13.081 [Thread-0] DEBUG net.ServerLogic - Removing client no. 1
```

Figure 4: Ausschnitt eines Logfiles

Bis zum 4. Meilenstein wollen wir folgende Ziele erreicht haben:

- Alle zentralen Code Stücke sind mit einem Logging versehen, welches die Fehlersuche vereinfacht.
- In regelmässigen code Reviews wird das Logging Tool angewendet.
- An mindestens einem Review wird aktiv das Logging Tool eingesetzt.

List of Figures