Unser Qualitätssicherungskonzept

Dieses Dokument soll zeigen, wie wir bestimmte Qualitätsaspekte erreichen wollen, die wir uns für unser Spiel „AmongAlien V2.0“ setzen. Das Ziel ist eine qualitativ hochwertige Software, die wir überprüfen und anpassen können. Dieses Konzept soll uns helfen, Fehler zu vermeiden und uns auf Schwierigkeiten aufmerksam zu machen.

Austausch und Absprache:

Damit jeder weiss, bis wann welcher Auftrag erledigt sein muss, haben wir ein Gant-Project, welches hierfür einen guten Überblick bietet. Desweitern haben wir einen Discord Server aufgesetzt, der mit verschiedenen Kanälen das Zusammenarbeiten massiv verbessert. Die Regel, stets im entsprechenden Channel zu sein, wenn man am Programmierprojekt arbeitet, hilft insofern, dass jeder stets weiss wann jemand anders am Programmierprojekt arbeitet und dazu stossen kann. In den verschiedenen nach Themen gegliederten Text-Chats findet man sehr leicht, was wozu besprochen wurde und die gepinnten Nachrichten, liefern zusätzlich Abhilfe.

Live Aktualisierung für unser Diary:

Damit jeder weiss, was wann und von wem wie gemacht wurde, führen wir ein Diary auf Google-Drive, da dort eine Live-Aktualisierung verwendet werden kann und wir so alle gleichzeitig daran arbeiten können, was diverse Vorteile hat und viele Problemquellen ausser acht lässt.

Übersicht im Bezug auf Methoden und Klassen:

Um sicherzustellen, dass wir einen lesbaren Code haben, haben wir die folgenden Regeln für die Codes in unserem Projekt:
Die Länge des Codes einer Methode wird kurzgehalten und wenn die Methode lang ist, wird sie in mehrere Methoden aufgeteilt. Wenn wir uns in Ausnahmefällen dafür entscheiden, die Methode lang zu halten, ist eine Erklärung erforderlich auch in Form von Kommentaren. Der Code sollte ausreichend dokumentiert sein.
Alle Methoden und Klassen sollten unterschiedliche Namen haben. Die gewählten Namen sollten auch die Funktion, dem Zweck oder das Objekt, das sie haben, so gut wie möglich widerspiegeln.
Eine Methode sollte mit einem Verb beginnen und aus bis zu drei Wörter bestehen. Das erste Wort beginnt mit einem kleinen Buchstaben und die folgenden Wörter mit einem Grossbuchstaben. Eine Klasse beginnt mit einem Substantiv und besteht aus bis zu zwei Wörtern. Alle Wörter beginnen mit einem Grossbuchstaben.

Verantwortungsbewusstsein:

Wir sprechen uns stets in der Gruppe ab und die Schnelleren unterstützen die Langsameren, sodass keine Arbeiten auf der Strecke bleiben. Wenn jemand etwas braucht, was in den Zuständigkeitsbereich eines anderen fällt, so wird mit diesem Kontakt aufgenommen und meist sofort gemeinsam daran gearbeitet oder die Erlaubnis erteilt, dies zu bearbeiten.
Infolgedessen muss keiner seine Arbeit pausieren und kann zeitnahe mit den benötigten Tools weiterarbeiten.

Sicherstellen der Funktionsweise und Testen:

Mit Unit-Test können wir Funktionen und Klassen einzeln testen. Dies wird erreicht durch eigene Testfälle für die einzelnen Klassen oder Funktionen geschrieben werden. Dadurch wird sichergestellt, dass der geschriebene Code richtig funktioniert. Wir werden bis zum fünften Meilenstein unsere Unit-Test erweitern. Die Javadoc sollte im gesamten Projekt so vollständig wie möglich sein.
Wenn ein Mitglied einen Fehler entdeckt, kann er ihn als Bug Report in unserem Discordchat stellen. Dazu gibt es einen extra Channel fehler-issue.

TODO's:

Die TODO's werden wenn immer nötig erstellt und im Discord-todo-Channel gepostet.

Sprache:

Innerhalb des Projekts werden Kommentare auf Englisch geschrieben. Auch Variablennamen sollten so gut wie möglich auf Englisch sein. Dies sorgt für einen einheitlichen und internationalen Code. Zudem macht es das ganze einfacher, das auch sonst alles was von Extern ist (Imports) auf Englisch dokumentiert ist.

Nützliche Vereinfachungen:

Angefangen bei IntelliJ, unserer Entwicklungsplattform mit Autocomplete Funktion, einem Refactor und vielen anderen nützlichen Tools verwenden wir diverse solcher Vereinfachungen. Um uns die Arbeit also nicht schwerer zu machen, als sie ist, greifen wir auf verschiedene Plug-Ins und Werkzeuge zurück. Mit dem Gradle-Build vereinfachen wir uns das Importieren externer Libraries. Weiter verwenden wir ein Checkstyle-Plug-In von IntelliJ, welches uns auf Fehler in der Darstellung des Programmcodes aufmerksam macht.

Durchführung:

Hier geht es darum zu sehen ob wir bei der Durchführung an unser Konzept gehalten haben.
Austausch und Absprache wurde immer gehalten. Das Gantt-Project war dabei hilfreich, jedoch war im Discord Server die Haupt Kommunikation. Dort haben wir Checklists und Fehler Kanäle sowieso TODO Kanäle welche sehr hilfreich bei der Strukturierung der Kommunikation waren. Die Idee der gepinnten Nachrichten war super, da man somit immer die wichtigsten Links auf Knopfdruck beis sich hatte.
Die Live aktualisierung der Diary erfolge immer nur teilweise, also jede paar tage wurde diese zusammen Aktualisiert, aber nicht immer täglich. Da könnte man besser gewesen sein, da man vor allem bei vieler Arbeit nicht mehr weiss, was wann gemacht wurde.

Die Javadocs sind sehr gut meistens immer beim erstellen der Methoden direkt auch erstellt worden. Somit hatten wir damit auch gegen Schluss wenige nicht dokumentierte Methoden.

Es waren alle Verantwortungsbewusst. Wir haben uns immer gegenseitig geholfen und waren immer Respektvoll zueinander. Bei hilfe wurde immer direkt Kontakt aufgeboten und wir haben sofort auf discord mit Bildschirmübertragung gearbeitet um probleme zu lösen.

TODOs wurden immer erstellt, diese holfen uns oft, da man bei vielem Code mit einer gelben TODO Markierung, den zu bearbeitenden Ort schnell finden kann.

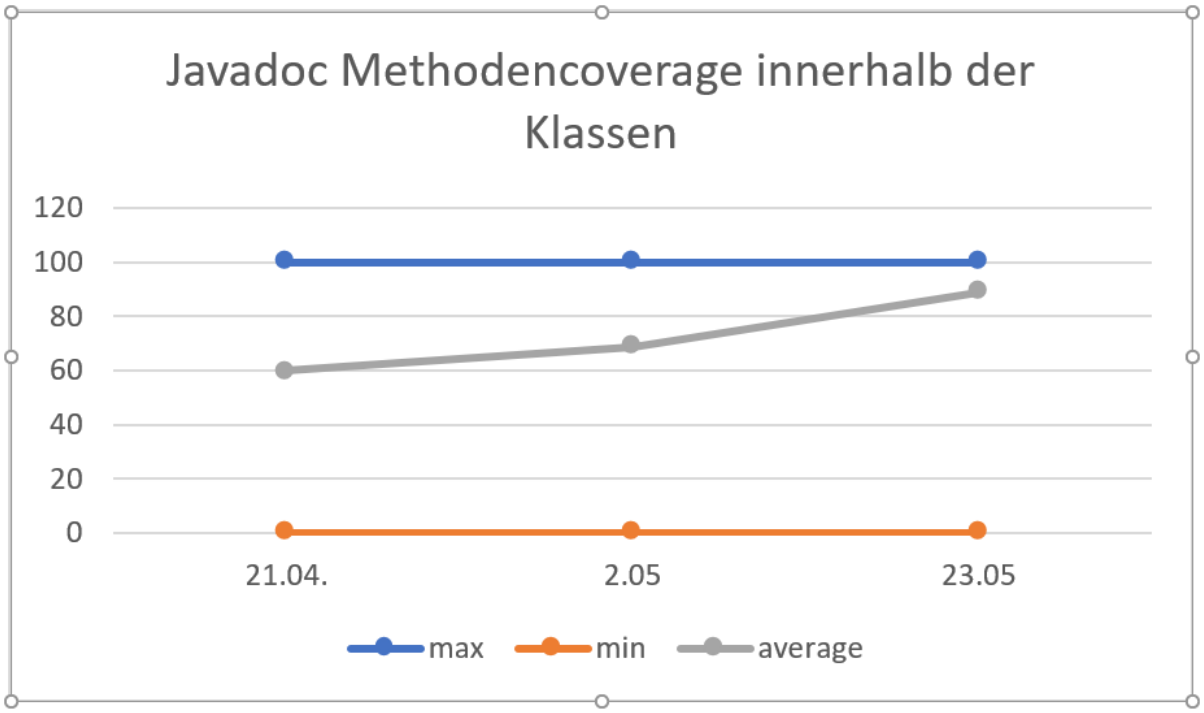
Hamachi wurde auch immer bei den meisten Tests gebraucht und dies lief immer sehr gut.
Die Englische Sprache zu nutzen war eine gute Idee und dies wurde auch von allen immer so übernommen und auch gebraucht.

IntelliJ hat viele funktionen und Plugins die verwendet haben sowie auch external Libraries, jedoch wollten einige sachen einfach nicht funktionieren wie Jacoco.

Resultate und Diskussion:

<<Javadoc Coverage>> der Methoden

Hier gibt es uns an wieviel Prozent der Methoden eine Javadoc haben und ist insofern praktisch, da man auf einen Blick sieht, wo man die Dokumentation noch vergessen hat und nicht manuell suchen muss. Der Test wurde an verschiedenen Meilensteinen ausgeführt:

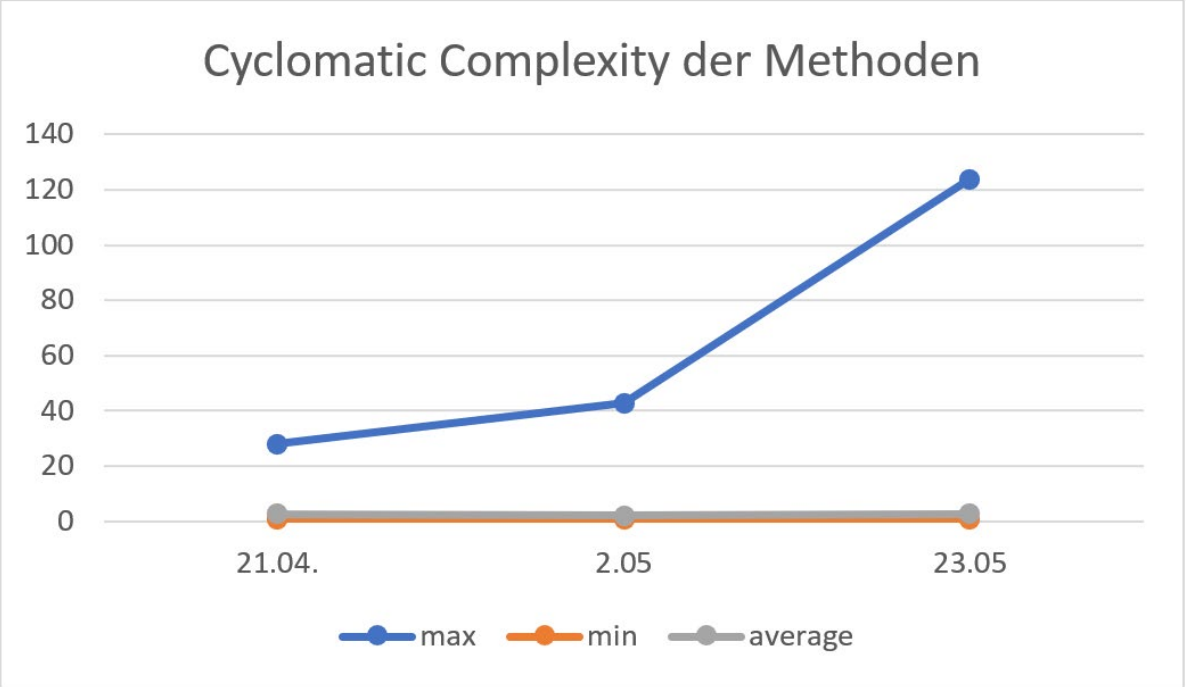


Man sieht, dass wir über Zeit immer besser geworden sind und mehr Methoden dokumentiert haben.

<<Javadoc Coverage>> der Methoden

Diese Metrik ist sagt aus, wie komplex eine Methode, ist. Je kürzer und simpler, desto besser. Dies erhöht die Programm Verständlichkeit und macht es einfacher, Fehler zu finden.
Metriken sind so definiert:

Cyclomatic Complexity unter 4: gut
Cyclomatic Complexity 5 - 7: mittel
Cyclomatic Complexity 8 – 10: hohe Komplexität
Cyclomatic Complexity über 10: extreme Komplexität



Die Komplexität wollen wir möglichst klein halten, damit das Programm einfacher zu verstehen ist.
Bei uns war es unter einer Komplexität von 3. Es war bei 2.56 average, was eine sehr Gute Komplexität ergibt.

<<Lines of Code>> pro Methode

Die Metrik hier sagt aus wie viel Lines of Code wir pro Methode haben. Je kürzer die Methode, desto verständlicher und leichter zu interpretieren ist es.
Deshalb haben wir uns bemüht kurze Methoden zu schreiben.
Jedoch gab es Methoden bei denen es zu sehr Langen Zeilen von Code kam wie man unten sehen kann. Dies kann man in der Zukunft verbessern indem man die Methode in unterteile aufspaltet um dort einzelne Aufgaben auszuführen.