Software Qualitätssicherung der Programmierung von Hexchess

1. Wir haben wöchentlich eins bis zwei Meetings als ganze Gruppe. In diesen Meetings werden wir den aktuellen Stand unserer Programme den anderen Mitgliedern kurz präsentieren:
   - Welche Funktionen bereits funktionieren.
   - Was noch zu tun ist,
   - Welche Schwierigkeiten vorhanden sind, die man selbst nicht lösen kann oder konnte.

2. In diesen Meetings sollten folgende Feedbacks gegeben sein:
   - Coding Stil evtl. auf Convention-Fehler Hinweisen.
   - Wenn nicht leserlich oder verständlich, soll die betreffende Person dies auch erfahren. So weiss sie, wie sie ihr Codestil anpassen kann.

3. Je nach Programm, kann dann mit Code With me gemeinsam programmiert werden, um z.B. Fehler zu beheben bzw. Lösungsansätze für bestehende Probleme zu finden oder es werden in den jeweiligen Gruppen weitergearbeitet.

4. Wir prüfen den Code der anderen auf:
   Lesbarkeit
   Reproduzierbarkeit
   Coding Conventions
   Usw.

5. Um einen Überblick zu behalten, sollten erwartete Werte in einer Tabelle (siehe File: QA Table) so gut wie möglich eingetragen werden. Dies trägt auch zur Verständlichkeit eines Codes bei. Ausserdem sollten Tests anhand dieser Tabelle erstellt werden können.

6. Aktuell ist es so, dass wir viel zu viel Code in einigen Klassen haben und auch, dass sich einige Codes wiederholen. Dies hat bereits dazu geführt, dass wir kaum einen Überblick haben. Ziel ist es, während Meilenstein 4 auch die bestehenden Codes an die folgenden Metriken anzupassen. Jedoch soll sicherlich dafür gesorgt werden, dass neue Klassen und Funktionen sich an die Metriken halten.

7. Wir haben uns für die folgenden drei Metriken entschieden:

    a. Anzahl Zeilen Code in einer Funktion: so sehen wir, ob eine Funktion (und meist auch gleich die entsprechende Klasse) «überladen» ist. Das soll uns einen besseren Überblick gewährleisten, was aber nicht bedeutet, dass zu viele Funktionen generiert werden sollen (kontraproduktiv). => max

    b. Anzahl Methoden-Aufrufe: so kann das Problem in a zum einen verhindert werden, zum anderen erkennen wir auch, wie «wichtig» unsere Methoden sind oder ob sie sowieso nur einmal verwendet wird. => min

    c. Übersicht javadoc: so können wir abschätzen, wie gut die Methoden kommentiert wurden. => average

8. Logger Libary soll zum Debuggen verwendet werden und unsere Outputs in einem Logfile speichern, so dass wir das Ergebnis wieder abrufen können, anstelle erneut zu testen. (hier mit Log4j2)

9. Wir basieren unsere Unit-Tests wie bereits erwähnt auf unsere Tabelle. Aber auch nicht eingetragene Funktionen werden wir bei Bedarf mit Unit-Tests testen. Für das Code Coverage wird Jacoco verwendet.

Metrik Messungen:

Messungen mit Libaries oder Jacoco wurden noch nicht durchgeführt.

Die möglichen Tests aus der Tabelle wurden manuell durchgeführt und auch nicht dokumentiert.

Zu 7c haben wir Einblick in alle Klassen:
Am Beispiel der Klasse Net.java kann man erkennen, wie gut die Methoden in dieser Klasse kommentiert wurden. Man sieht zum Beispiel, dass es vereinzelte Methoden gibt, die nicht beschrieben wurden. Man kann also die betreffende Person darauf hinweisen, dass sie ihre Methode per javadoc kommentieren muss. Als Überblick der Klassen, ist diese Metrik optimal.

Constructor	Description
`Net()`	

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	`broadcast(java.lang.String name, java.lang.String msg)`	needed for achievement
static void	`chooseLobby(int i, java.lang.String name)`	adds player to the chosen Lobby if possible
static void	`command(char[] packet)`	
static void	`createLobbies()`	creates empty lobbies
static void	`error(char[] packet)`	Sends Error
static void	`getList(java.lang.String name, java.lang.String msg)`	prints list of lobbies, usernames or game status to this specific user
static void	`lobbyBroadcast(java.lang.String msg, Lobby lobby)`	sends message to all lobby players (max 3)
static void	`refresh(char[] packet)`	
static void	`rename(java.lang.String name, java.lang.String input)`	changes username if entered username is unique
static void	`sendBel(char[] packet, java.lang.String name)`	
static void	`sendMsg(java.lang.String name, java.lang.String msg)`	send message to all Clients
static void	`sendMsgBack(java.lang.String name, java.lang.String msg)`	sends message back to this user, as in feedback
static int	`validate(char[] packet)`	