Software-Qualitätssicherung im Projekt “Rat um Rad”

Massnahmen zur Qualitätssicherung

Checkliste für Merge-Requests, um eine ausgezeichnete Qualität des Code zu gewährleisten.

- [x] New classes and methods have Javadoc.
- [x] Testable Methods and Classes are tested.
- [x] Project builds (Pipeline).
- [x] All Tests pass (Pipeline).
- [x] Approved and reviewed by a team member.
- [x] Updated documentation documents if needed.
- [x] Code is readable and has no unused code or commented out code.
- [x] Code is properly formatted (Ctrl+Alt+L).

Dokumentation mittels Javadoc (& nach Javadoc Konventionen)
Einsatz von UnitTests für testbare Komponenten
Code Review durch Team Member
befolgen der Coding Konventionen

Testing

Es wird angestrebt, alle testbaren Bereiche des Spiels zu testen.
Während der Entwicklung werden Unit-Tests (mit JUnit) geschrieben, um die einzelnen Methoden möglichst durchgehend zu testen. Nach Abschluss einer Teilaufgabe wird die Funktionalität zudem mit manuell durch die Entwickler getestet (Nach Black- und Whitebox Verfahren).

Ein besonderes Augenmerk werden wir auf das Player-Management legen. Dies tun wir, da das Management der Player sich sowohl über viele Teilbereiche der ganzen Software zieht (Connection herstellen, Player-Management während des Spiels, sinnvolles Connection-Loss-Handling während des Spiels, Ping-Pong, etc.) als auch, weil eine sinnvolles Connection-Loss-Detection und -Handling anspruchsvoll sein kann (z.B. Weitergehen des Spiels trotz bei Verbindungsstörungen sichern, Reihenfolge der Aufgaben bei Verbindungsverlust).

Dokumentation für manuelle Tests: Dedizierten Testordner mit Template für Tests auf Google Drive. Ablage auf Gitlab zu jedem Meilenstein.

Logging

Für das Logging verwenden wir Log4j. Prozesse sollen in separaten Logfiles geloggt werden.
Zum jetzigen Stand gibt es für das ConnectionHandling (Achievement Ping und Pong) ein eigenes Logfile (aktuell als custom Level definiert*). Weitere geloggten Warnungen oder Debugnachrichten werden in einem allgemeinen Logfile geloggt. Zudem unterscheiden wir zwischen den unterschiedlichen Level und geben z.B. Error-Logs zusätzlich in der Commandline aus.

So soll eine eine gezielte Fehlersuche in unterschiedlichen Bereichen und von unterschiedlichen Personen (Anwender*innen, Entwickler*innen) ermöglicht werden. Zudem wird unterschieden zwischen Nachrichten, die für den User wichtig sind, und Nachrichten, die für die erweiterte Fehlersuche nötig sind.

* Es ist zu entscheiden, ob ein custom Level auch in Zukunft verwendet werden soll, oder die Umsetzung durch Marker sinnvoller ist.

Teamkultur

Kommunikation mit Respekt, Fehlertoleranz und gemeinsamer Verantwortung.
Förderung der Teamkultur durch Pair Programming (mit IntelliJ “Code with Me”)
Code Reviews zur Qualitätssicherung (z.B. Merge-Requests auf Gitlab reviewen) nach Checkliste aus Dokument 06-cs108-FS23-SW_Qualitaetssicherung auf der Seite 36.
Definierte Prozesse (Regelmässige Sitzungen, Zeitplan, Aufgabenteilung → Projektplan, Tagebuch)

Bug Reports: Falls Bugs während des Code Review oder während des Ausprobierens auffallen.
Template auf Google Drive hinterlegen und Bug Report jeweils an zuständige Entwickler weiterleiten.

Messungen

Einsatz von Jacoco, um die Code-Coverage zu überprüfen.
Anzahl Logging-Statements im Verhältnis zu Lines of Code
3 frei wählbare Metriken: *(pro Metrik Minimum, Maximum und Durchschnitt)*

1) Anzahl von mit Javadoc dokumentierten Methoden (Lines of Javadoc)
2) Test Coverage (zu bestimmen wie zu messen)
3) Lines of Code pro Methode (eigentlich besser, jede Methode macht nur 1 Aufgabe) → noch zu bestimmen wie messen