Software-Qualitätssicherung

Product Management

Für unser Projektmanagement verwenden wir OpenProject. Dort nutzen wir Timelines, um den Fortschritt zu visualisieren, und organisieren die Aufgabenteilung direkt innerhalb der Timelines. Dies stellt sicher, dass alle Teammitglieder den aktuellen Entwicklungsstand und ihre jeweiligen Aufgaben kennen.

Zusätzlich setzen wir auf regelmässige Meetings, um den Entwicklungsprozess effizient zu steuern und mögliche Probleme frühzeitig zu erkennen.

Versionskontrolle & Code-Management

Unsere Versionskontrolle erfolgt über GitLab. Um eine saubere und strukturierte Entwicklung sicherzustellen, wollten wir ein Branching-Modell verwenden. Dadurch kann jedes Teammitglied ungestört an einzelnen Features oder Bugfixes arbeiten, bevor die Änderungen in den Hauptzweig integriert werden. Leider konnten wir dies aber nicht effizient umsetzen.

Jede Änderung wird erst nach Code-Reviews und Merge Requests in den Hauptzweig übernommen, um Fehler frühzeitig zu erkennen und die Code-Qualität zu gewährleisten.

Ausserdem verwendeten wir die Build-Pipeline auf Gitlab, um sicherzustellen, dass jeder Commit korrekt gebuildet wurde. So konnten wir den Prozess gut im Überblick behalten und Fehler direkt erkennen.

Code-Reviews & Testing

- Code-Reviews: Jedes Feature oder jeder Fix wird vor dem Merge von mindestens einem weiteren Teammitglied überprüft. Dies verbessert Code-Qualität, Lesbarkeit und Wartbarkeit.
- Automatisierte Tests: Wir haben Unit-Tests für kritische Spielmechaniken implementiert und konnten Fehler frühzeitig erkennen.
- Manuelles Testing: Zusätzlich testen wir regelmässig Gameplay, UI und Performance, um ein optimales Spielerlebnis sicherzustellen.
- Coding-Style: Wir verwenden eine angepasste Version von Google-Style für Java. Wir versuchen, Verschachtelungen zu vermeiden, um die Code-Lesbarkeit zu gewährleisten.

Tools

Um die Code-Qualität weiter zu verbessern, planen wir folgende Tools einzusetzen:

MetricsReloaded

- Für die Codeanalyse verwenden wir das Plugin **MetricsReloaded** direkt in IntelliJ IDEA.
- Es liefert Metriken zur Wartbarkeit, insbesondere in Bezug auf die Grösse, Komplexität und Kopplung einzelner Klassen und Methoden.
- Dadurch lassen sich potenzielle Designprobleme frühzeitig erkennen und die langfristige Code-Qualität verbessern.

slf4j

- Zum Logging verwenden wir slf4j mit log4j als Backend, um eine detaillierte Fehleranalyse zu ermöglichen.
- Die Logs werden in der Konsole ausgegeben, um zur Fehlerdiagnose genutzt werden zu können, z. B. zur Überprüfung, ob die Protokoll-Kommandos korrekt gesendet und empfangen wurden.
- Dies erleichtert das Debugging und die Nachverfolgbarkeit von Fehlern.

Messungen

	Lines of code - classes					
	Total			Average		
Milestone	Comments	Javadoc	Code	Comments	Javadoc	Code
Milestone 2	306	248	1204	30.60	24.80	120.40
Milestone 3	1072	966	4623	53.60	48.30	231.15
Milestone 4	1343	1134	7794	38.37	32.40	222.67
Milestone 5	1622	1406	10539	30.60	26.53	198.85

Figure 1: Lines of code: Classes

	Lines of code - methods					
	Total			Average		
Milestone	Comments	Javadoc	Code	Comments	Javadoc	Code
Milestone 2	225	204	1102	4.33	3.92	21.19
Milestone 3	757	826	4227	3.79	4.13	21.14
Milestone 4	946	952	7276	3.25	3.29	25.18
Milestone 5	1181	1224	9828	2.60	2.71	21.74

Figure 2: Lines of code: Methods

	Complexity - classes			
	Average			Total
Milestone	Average operation	Maximum operation	Weighted method	Weighted method
Milestone 2	2.68	6.20	16	158
Milestone 3	3.06	8.39	30.80	616
Milestone 4	3.02	6.04	26.83	805
Milestone 5	2.73	5.32	24.13	1279

Figure 3: Complexity: Classes

	Complexity - methods					
	Total			Average		
Milestone	Cognitive	Design	Cyclomatic	Cognitive	Design	Cyclomatic
Milestone 2	202	162	83	3.67	3.12	1.60
Milestone 3	887	639	746	4.41	3.19	3.73
Milestone 4	1226	819	986	4.59	3.08	3.70
Milestone 5	1807	1375	1563	3.86	3.04	3.46

Figure 4: Complexity: Methods

Diskussion

Anzahl Code-Zeilen

Insgesamt zeigt sich eine positive Entwicklung über die Meilensteine hinweg. Der Codeumfang nimmt kontinuierlich zu, was auf Fortschritte in der Implementierung und eine wachsende Funktionalität hinweist. Auch die Anzahl an Kommentaren und Javadoc-Einträgen steigt, was grundsätzlich auf eine gewisse Dokumentationsdisziplin schliessen lässt.

Allerdings lässt sich anhand der Durchschnittswerte erkennen, dass einzelne Klassen und Methoden teilweise sehr gross geworden sind. Dies kann langfristig zu Wartbarkeitsproblemen führen. Auch der rückläufige Durchschnitt an Kommentaren und Javadoc pro Methode bzw. Klasse deutet darauf hin, dass mit zunehmendem Umfang weniger dokumentiert wird.

Erkenntnisse

Es wäre daher empfehlenswert, in zukünftigen Projekten auf eine bessere Strukturierung durch kleinere, übersichtlichere Klassen und Methoden sowie eine konsequentere Dokumentation zu achten.

Komplexität

Die durchschnittliche Komplexität der Klassen blieb über die Milestones hinweg weitgehend stabil. Der Rückgang in Milestone 5 deutet auf bewusste Strukturierungsmassnahmen hin. Auch die maximale Komplexität einzelner Methoden wurde reduziert, was auf gezieltes Refactoring schliessen lässt.

Auf Methodenebene zeigt sich ein leichter Anstieg der cyclomatischen Komplexität, allerdings bleibt diese im akzeptablen Bereich. Kognitive und Designkomplexität sind über alle Milestones hinweg relativ konstant und deuten auf eine gute Lesbarkeit des Codes hin.

Erkenntnisse

In zukünftigen Projekten sollte darauf geachtet werden, Methoden mit hoher cyclomatischer Komplexität frühzeitig zu identifizieren und gegebenenfalls durch Aufteilung zu vereinfachen. Eine weitere Reduktion stark gewichteter Klassen würde ebenfalls zur langfristigen Wartbarkeit beitragen.

Fazit

Insgesamt zeigt sich, dass das Team während der Projektumsetzung wichtige Aspekte der Software-Qualität adressiert hat. Durch strukturierte Code-Reviews, den Einsatz von Metrik-Tools sowie automatisierte und manuelle Tests wurde eine solide Grundlage für wartbaren und nachvollziehbaren Code geschaffen.

Die Analyse der Codezeilen und Komplexitätsmetriken zeigt eine kontinuierliche Erweiterung des Projekts bei gleichzeitig stabiler oder sogar sinkender durchschnittlicher Komplexität. Dies spricht für eine bewusste Strukturierung und Qualitätssicherung im Entwicklungsprozess.

Gleichzeitig wurden Herausforderungen sichtbar, insbesondere in Bezug auf zu grosse Klassen und Methoden sowie rückläufige Dokumentation pro Einheit. Diese Punkte sollten in zukünftigen Projekten gezielt adressiert werden, um langfristig eine hohe Wartbarkeit und Verständlichkeit sicherzustellen.

Durch die gewonnenen Erkenntnisse ist das Team gut vorbereitet, künftige Projekte noch effizienter und qualitätsbewusster umzusetzen.