Abstract
Wir haben ein Schachspiel in Java basierend auf Swing entwickelt. Fertig implementierte Funktionen sind das lokale Spiel, der Mehrspielermodus (Peer-to-Peer, ohne zentralen Server), die Speicherung und das Laden eines Spielstands, eine Undo/Redo-Funktion sowie ein Live-Log der Spielzüge. Die Architektur wurde modular nach dem MVCS-Prinzip aufgebaut, wodurch die Wartbarkeit und Erweiterbarkeit vereinfacht werden.

Projektidee
Ziel war die Entwicklung eines Schachspiels, das sowohl lokal als auch über Netzwerk spielbar ist. Spieler sollen Partien speichern, laden und Züge zurücknehmen können.

Anforderungsdefinition

Must-Have:

- Lokales Schachspiel mit GUI
- Spiellogik (reguläre und spezielle Züge)
- Undo/Redo
- Speicherung und Laden von Spielständen
- Peer-to-Peer-Multiplayer (Host/Client)

Nice-to-Have:

- Anpassbare Figuren-Designs
- KI-Gegner
- Online-Matchmaking / Ranking

Architektur & Struktur
Die Architektur orientiert sich am MVC-Prinzip und wird durch ein Service Package ergänzt (MVCS).

- Model: Enthält die Spiellogik und Datenstrukturen. Hier werden das Brett, die Figuren und der aktuelle Spielzustand verwaltet. Alle Schachregeln werden in diesem Layer umgesetzt, inklusive Sonderregeln wie Rochade oder en passant. Das Model ist unabhängig von Oberfläche und Netzwerk und kann einzeln getestet werden.
- View: Stellt die grafische Benutzeroberfläche mit Swing bereit. Dazu gehören das Schachbrett, Menüs, Buttons, Statusanzeigen und die Zugliste. Die View zeigt ausschließlich den Zustand des Models an und reagiert visuell auf Änderungen.
- Controller: Steht zwischen Benutzereingaben, View und Model. Interpretiert Mausklicks und Menüaktionen, prüft Züge über das Model, aktualisiert die View und koordiniert im Mehrspielermodus die Kommunikation mit den Netzwerk-Services. Der Controller gibt Aufgaben weiter an Model und Services.
- Services: Übernehmen unterstützende Aufgaben außerhalb der Kernlogik. Dazu gehören das Speichern und Laden von Spielständen, die Undo/Redo-Verwaltung (MoveHistory) sowie die Netzwerkkommunikation (Host/Client über TCP).

Sie erweitern die Spielfunktionalität, ohne dass Model oder Controller verändert werden müssen.

Diese Aufteilung erlaubt uns beispielsweise die Logik und GUI unabhängig voneinander zu entwickeln und vereinfacht eine potenzielle Erweiterung.

Konzeption & Projektplanung

- Must-Have: Lokales Spiel, GUI, Mehrspielermodus (Host/Client), Undo/Redo, Speichern & Laden
- Should-Have: Menü mit Hintergrundgrafik, Move-Log
- Nice-to-Have: KI, alternative Figurendesigns, Ranking/Matchmaking

Stand des Projekts

- Fertig: Lokales Spiel, GUI, Undo/Redo, Multiplayer (Host/Client), Speicher-/Ladefunktion.
- Nicht umgesetzt: KI-Gegner, alternative Figurendesigns, Matchmaking.
- Macken:
  - Verbindung über Internet nur möglich, wenn Host einen offenen Port bereitstellt
  - Peer-to-Peer-Verbindung ist unverschlüsselt

JUnit Tests
Wir testen vor allem die Erstellung des Schachbretts, Positionen, Spielzüge, Schacherkennung, Undo/Redo. Die Tests konzentrieren sich auf die Logik im Model/Controller.

Abgedeckte Testklassen und Inhalte:

- BoardFactoryTest: prüft die Anfangsaufstellung der Spielfiguren und stellt sicher, dass Felder in der Mitte des Schachbrettes bei der Startstellung leer sind.
- PositionTest: testet Wertgleichheit identischer Koordinaten und wirft bei ungültigen Koordinaten (außerhalb 0–7) eine Exception.
- PawnPseudoMovesTest: verifiziert beim weißen und schwarzen Bauern vom Startfeld die korrekten Vorwärtszüge (ein bzw. zwei Felder), wenn der Weg frei ist.
- CheckDetectionTest: stellt sicher, dass die Startposition nicht im Schach ist, erkennt Schach über freie Linien (Turm >> König) und kein Schach bei blockierter Linie.
- GameStateCopyTest: prüft, dass GameState.copy() eine korrekte Kopie liefert (Zug im Kopie-Zustand verändert das Original nicht).
- UndoRedoControllerTest: testet Funktion von Undo/Redo am Controller über die API.

Workload
Der tatsächliche Arbeitsaufwand lag über der vorgesehenen Zeit. Wir haben viel zusätzliche Zeit, vor allem außerhalb der Uni, investiert, um alle Funktionen rechtzeitig zu implementieren. Die Probleme, die wir außerhalb der Programmierung hatten (siehe „Probleme und Selbstreflexion“, haben den Zeitaufwand zusätzlich in die Höhe getrieben. Insgesamt lag der Zeitaufwand mit der Planung, Dokumentation, reinen Programmierung, Fehlerbehebung, Git-Verwaltung/Probleme und dem Testen bei etwa 100 Stunden.

Aufteilung
Jeremy hat hauptsächlich an den Spiellogiken (normale und spezielle Züge) gearbeitet, während Sebastian für das Frontend (Java Swing) und die Zusatzfunktionen, wie Undo/Redo oder die Netzwerk-Implementation zuständig war.

Was wurde weggelassen?

- KI-Gegner: zu komplex und zeitaufwändig für den gegebenen Rahmen
- Alternative Skins für Figuren: gestrichen zugunsten von Kernfunktionen
- Matchmaking und Bewertungssystem: nicht realisierbar ohne großen Mehraufwand

Probleme und Selbstreflexion
Während der Entwicklung gab es zwei wesentliche Herausforderungen:

1. GitHub-Merge: Jeder arbeitete auf einem eigenen Branch. Beim Zusammenführen in der Uni traten regelmäßig Konflikte auf, was vor allem anfangs viel Zeit kostete.
2. Überschätzung des Projektumfangs: Unsere ursprüngliche Planung enthielt deutlich mehr Features, darunter auch eine KI als Gegner. Diese war in der vorgegebenen Zeit nicht realisierbar. Wir mussten Prioritäten neu setzen und uns auf die Kernfunktionen konzentrieren.

Mögliche Erweiterungen
Wenn wir mehr Zeit gehabt hätten, hätten wir gerne weitere Features implementiert, um das Projekt noch umfangreicher und interessanter zu gestalten.
Dazu gehört vor allem die Implementation eines KI-Gegners mit verschiedenen Schwierigkeitsstufen, um auch allein gegen den Computer spielen zu können. Außerdem hätten wir gerne ein Rankingsystem auf Basis von Elo eingebaut, kombiniert mit einem Multiplayer-Matchmaking, sodass Spieler automatisch gleichstarke Gegner finden.
Für den Mehrspielermodus könnte zudem ein sicheres, verschlüsseltes Netzwerkprotokoll (TLS) umgesetzt werden. Ein weiterer Zusatz wäre ein Chat im Netzwerkmodus, damit die Spieler während der Partie direkt kommunizieren können. Schließlich hätten wir Farb- und Designmöglichkeiten für Schachbrett und Figuren eingebaut, um die GUI optisch anpassbarer und individueller gestalten zu können.