Praxis der Softwareentwicklung

Implementierungsbericht

Rukiye Devran, Tim Groß, Daniel Helmig, Orkhan Aliev

Inhaltsverzeichnis

1 Ablauf

Es wurde weitgehend unabhängig voneinander implementiert mit einigen gemeinsamen Treffen um den Fortschritt zu begutachten und eventuelle Differenzen zwischen Komponenten zu beseitigen. Es gab im ins gesamten drei Komponenten: Die Schachlogik, Server und Benutzeroberfläche. Die Schachlogik von Tim Groß, der Server von Daniel Helmig und Benutzeroberfläche wurde von Rukiye Devran implementiert. Orkhan Aliev hat bei der Schachlogik und bei dem Server mitgeholfen, da diese aufwändig waren.

2 Schwierigkeiten

2.1 Allgemein

Teilweise gab es Unstimmigkeiten zwischen den verschiedenen Komponenten Benutzeroberfläche, Schachlogik und Server, bei denen nicht ganz klar war welche Funktionalität bereit gestellt wird und was im Laufe der Implementierung geändert werden musste und somit nicht mehr zur Verfügung steht.

2.2 Benutzeroberfläche

Zum einen war es sehr schwierig die Berechnungen auf dem Display zu implementieren, da man für eine gegebene Pixel-Position abhängig von der Größe des Displays und der Position des Schachbrettes die richtige Kachel berechnen musste. Außerdem war es kompliziert mit den vielen ImageView auf dem Schachbrett umzugehen, da man im Layout alles richtig constrainten und die Attribute korrekt festlegen musste, damit anschließend bei der Ausrichtung auf dem Schachbrett alles funktionierte. Desweiteren war es umfangreicher als erwartet die Bauerntransformation zu gestalten. Es war auch sehr schwierig das Funktionsprinzip der benutzen Grafikeinheiten zu verstehen, da diese teilweise sehr abstrakt und kompliziert sind. Allgemein gesagt, war es umfangreicher und komplizierter die Benutzeroberfläche zu implementieren, da mehr Funktionalität als erwartet in der Benutzeroberfläche vonnnöten war.

2.3 Schachlogik

Im Laufe der Implementierung erwiesen sich einige Klassen als überflüssig. Andere Funktionalitäten fehlten im Entwurf und mussten zusätzlich implementiert werden.

2.4 Server

Ein Hindernis war die Umstellung von Javalin mit eingebettetem Server zu einem Web-Servlet. Diese Funktionalität ist neu in Javalin und hat zu viele Probleme verursacht, sodass der gesamte ClientHandler neu geschrieben wurde. Dieses mal mithilfe des Spring Frameworks.

Probleme gab es zudem beim Testen des Servers aufgrund nicht optimalen Entwurfs. Die Klassen GameCreator, MoveHandler und BoardHandler bekamen als Parameter nur den/die Spieler beziehungsweise den Zug. Daher bekamen alle drei den DatabaseHandler direkt über die statische Methode getHandler. Dies erschwerte das Testen dieser Klassen ungemein und wurde gelöst indem der DatabaseHandler nun im Konstruktor übergeben wird.

Generell erwies sich das Testen des Servers als schwierig, da sehr viele externe Bibliotheken verwendet wurden und viele statische Aufrufe gemacht werden.

3 Veränderungen am Entwurf

3.1 Benutzeroberfläche

Der GUI-Entwurf wurde soweit übernommen, jedoch gab es auch einige Veränderung:
Zusätzlich zu der **BoardActivity**, **MainMenuActivity**, **StatisticsActivity** und **SearchPlayerActivity** kam noch weitere Activities dazu:
Für die Bauern-Transformation wurde noch eine **PawnActivity** erstellt, die meisten Dialoge die im Entwurfsheft standen wurden alle in Activities umgewandelt, d.h. es gibt jetzt eine **ChallengeActivity**, **DrawActivity**, **LoginActivity**, **LostActivity**, **WinnerActivity**.
Zusätzlich zu der **SearchPlayerActivity** wurde eine Hilfsklasse **SearchPlayerListViewAdapter** erstellt.
In der BoardActivity kamen noch zusätzliche Methoden hinzu:

- **boardClicked(MotionEvent event):**
  Wenn das Schachbrett angeklickt wird, berechnet die Mehtode die angetippte Kachel.

- **moveFigure(ImageView iv, Position targetPosition, int animationDuration):**
  Bewegt die Figuren auf dem Schachbrett.

- **clearColors():**
  Entfernt die Färbung von dem möglichen Zug.

- **colorMoves(List<Move> moves):**
  Färbt alle möglichen Züge ein, indem eine Bitmap benutz wird.

- **ImageView getPieceIV(String pieceRepresentation, int ivCounter):**
  Sucht ein unbenutztes ImageView und ordnet diesem das Bild der angefragten Figur zu und gibt es dann zurück.

- **paintBoard(BoardState board):**
  Bewegt die ImageViews mit den entsprechenden Bildern auf die von einem übergebenen BoardState vorgegebenen Felder.

  Die **SearchPlayerActivity** wurde komplett geändert.
  Es gibt eine Methode onCreateOptionsMenu, welche eine Liste generiert und von der Hilfsklasse SearchPlayerListViewAdapter einen Filter von der Mehtode filter() bekommt.
  In der **StatisticsActivity** wurde keine Methode verwendet. In dieser Klasse werden die Daten nicht wie Anfangs geplant auf dem Server gespeichert, sondern lokal auf dem Smartphone gespeichert.
  Bei der **LoginActivity** speichert man nur den Spielernamen lokale auf dem Smartphone. Die Google-Anmeldung wurde verworfen, da der Spieler die Möglichkeit haben soll, einen eignen Spielernamen zu wählen und wir selber eine Anmeldung gestalten wollten.

3.2 Schachlogik

3.2.1 Hinzugefügte/Entfernte Klassen

-Game
  Das Konzept eines Spiels als eigenes Objekt wurde komplett verworfen, da vom Server sowie von der GUI lediglich das Brett und der Regellieferer separat verwendet werden.

-Player
  Es ist nicht nötig, Spieler zu modellieren, da das Konzept eines Spiels verworfen wurde. Spieler werden anhand ihres Namens erkannt, und so vom Server abgerufen.

-User
  Daten wie gewonnene/gespielte Spiele werden direkt in der App gespeichert. Über

den Namen eines Spielers kann dieser direkt vom Server abgefragt werden, so kann eine eigene Modellierung umgangen werden.

+MoveFactory
Das Erstellen eines Zuges anhand eines eindeutigen Strings, den man durch die toString-Methode erhält, ist an mehreren Stellen notwendig. Da es jedoch mehrere Unterklassen gibt, welche von Move erben, kann nicht einfach ein allgemeiner Konstruktor verwendet werden, welcher den String entgegen nimmt. Es ist eine eigene Klasse, eine sogennante Fabrik, notwendig, um den jeweiligen Typ von Zug anhand des Strings erstellen zu können. Die getMove-Methode der MoveFactory gibt den korrekten Typ von Zug zurück, bei fehlerhaften Argumenten wird null zurückgegeben.

+PieceFactory
Wie bei Move muss auch eine Figur an mehreren Stellen anhand ihrer eindeutigen String-Repräsentation erstellt werden können. Da auch Piece eine Vererbungshierarchie aufweist, welche einen universellen Konstruktor unmöglich macht, wird auch hier eine Fabrik benötigt. Die getPiece-Methode der PieceFactory erstellt eine Figur mit ihrer korrekten Farbe und gibt diese zurück, bei falschen Argumenten wird null zurückgegeben.

3.2.2 Hinzugefügte/Entfernte/Veränderte Methoden

RuleProvider
Die Methode isChecked(boolean, BoardState) wurde private gemacht, da sie nie von außerhalb benutz werden muss.

Die private Methode hasLegalMoves(BoardState) wurde hinzugefügt, diese prüft ob der ziehende Spieler mögliche Züge hat. Dies wird zur Überprüfung auf Matt/Patt benötigt.

Die neue private Methode isAllowedMove(Move, BoardState) prüft, ob ein Zug auf dem Brett erlaubt ist. Dazu wird nur geprüft, ob der eigene König nach dem Zug im Schach stehen würde. Diese Methode wird von getLegalMoves(Position, BoarsState) zum Filtern der von getPossibleMoves(Position, BoardState) übergebenen Züge verwendet. Die öffentliche Methode isLegalMove(Move, BoardState) prüft zusätzlich, ob an der Startposition des Zugs überhaupt die richtige Figur steht und außerdem ob der ziehende Spieler auch am Zug ist. Diese Methode wird vom Server zur Überprüfung auf Korrektheit verwendet.

BoardState
Der Konstruktor ohne Parameter wurde entfernt. Dieser sollte ursprünglich ein

Schachbrett in Startkonfiguration zurückgeben, diese Funktionalität wird aber bereits von der getStartState()-Methode des ChessRuleProviders implementiert.

Es wurde die Methode hasPieceAt(Position) hinzugefügt, welche prüft ob an der gegebenen Position eine Figur steht. So werden null-Abfragen nach der Benutzung von getPieceAt(Position) vermieden, was den Code sauberer macht.

Die neue Methode getPiecesOfColor(boolean) ermöglicht schnelles iterieren über alle Figuren einer bestimmten Farbe, was zum Beispiel zur Überprüfung auf Schach nützlich ist.

Auch getKingOfColor(boolean) ist wichtig für die Überprüfung auf Schach. Es wird keine Fehlermeldung ausgegeben, wenn die Anzahl der Könige einer Farbe ungleich eins ist, da solche Stellungen durch das Ausführen legaler Züge nicht erreicht werden können, nur durch die gezielte Erstellung falscher Bretter mit Strings.

Die Klasse Tile erhielt eine neue Methode removePiece(), welcher die Figur auf diesem Feld auf null setzt (eleganter als setPiece(null)).

Result

Die toString()-Methode wurde in getResult() umbenannt.

Der Konstruktor mit nur einem Parameter wurde aufgrund fehlender Nutzung entfernt.

Move

Es wurde die Methode equals(Move) hinzugefügt, welche den jeweiligen Zug mit dem übergebenen vergleicht, was an vielen Stellen nützlich ist. Dazu werden einfach die Start- und Zielpositionen der Züge verglichen. Es ist nicht nötig ob es die gleichen Typen von Zug sind, da auf einem gegebenen Brett immer nur ein Zug mit gleichem Start und Ziel existiert.

Die to-String()-Methode wurde bei den Spiezielzügen verändert um die Strings einheitlicher zu halten, es wird nur noch Start, Ziel und ein Zeichen als Zusatz zurückgegeben, die zusätzlichen Informationen können aus diesen errechnet werden.

Die Konstruktoren von Position fangen nun fehlerhafte Eingaben ab und geben entsprechende Fehlermeldungen aus, ohne die Position zu erzeugen.

Positionen haben nun eine equals(Position)-Methode, welche zwei Positionen anhand ihrer Koordinaten vergleicht.

Piece

Die toString()-Methode der jeweiligen Figuren beachtet nun die Farbe der Figur,

und gibt bei schwarzen Figuren den entsprechenden Kleinbuchstaben zurück.

Es wurde eine Methode getImageName() für alle Figuren hinzugefügt, welche den Namen des jeweiligen Bildes zurückgibt, das diese Figur auf dem Schachbrett repräsentieren soll.

3.3 Server

3.3.1 Hinzugefügte/Entfernte Klassen

SocketHandler und WebSocketConfig Beide Klassen ergaben sich aus der implementierung mit Spring.

3.3.2 Hinzugefügte/Entfernte/Veränderte Methoden

ClientHandler Der ClientHandler wurde um folgende REST-Schnittstellen erweitert:

- **/delete/player:** Um ein Spiel aus der Datenbank zu löschen
- **/players:** Um alle Spieler seit Server Start abzufragen
- **/isonline/player:** Um abzufragen ob ein Spieler online ist
- **/hasgame/player:** Um abzufragen ob ein Spieler bereits in einem Spiel ist