

Programmentwurf Tim-Robin Kalkhof

Inhaltsverzeichnis

Programmentwurf Tim-Robin Kalkhof	1
Idee.....	2
Unit Tests.....	3
Clean Architecture	4
Refactoring	5
Programming Principles.....	6
Entwurfsmuster	7
Domain Driven Design	8

Idee

Visuelles Schachprogramm und Schachbot

GUI-Programm, dass es ermöglicht Schach gegen einen 2. Menschen vor dem PC oder einen Schachbot zu spielen. Außerdem verschiedene Schachuhr Presets, für verschiedene Zeiteinstellungen. Hierbei wird die komplette Logik des Spieles Schach implementiert. Außerdem werden Negamax und Alpha-Beta Pruning Algorithmen verwendet, um einen Schachbot zu implementieren.

Möglichkeit gegen anderen Menschen oder den Computer Schach zu spielen.

Hierbei verwendete Technologien: Java, Swing/FX, Junit, AssertJ, Mockito, Gradle

Unit Tests

Beim Testen der Square Klasse war eine große Priorisierung der Aspekt, dass die Tests independent sein müssen.

Um die Abhängigkeiten bei der Image Loader Klasse zu reduzieren habe ich mocks verwendet.

Bei der Code Coverage habe ich besonders den Wert auf die Branch Coverage gelegt, anstatt nur die Coverage Prozentzahl zu priorisieren.

Clean Architecture

Für die Clean Architecture ist die Plugin Schicht das Swift UI und die Test Frameworks AssertJ und Mockito für Mocks. Weiterhin die Entities sind die Pieces auf der Domain Ebene .

Abstraction Ebene vielleicht alpha Beta?

Als Abstraction Ebene habe ich den Negamax Algorithmus bestimmt, da dies auf einem Mathematischen Prinzip beruht und immer von Grundauf korrekt ist.

Auf der Application Ebene ist dann die Board Klasse vertreten weiterhin auch GUI und Game

Adapters Board?

Als Adapters Klasse kann man die Board Klasse nennen, da diese Die Logik zwischen Den Figuren und dem GUI darstellt.

Refactoring

Die Init Methode in der Board Klasse ist sehr Lang und Komplex und enthält viele Duplikate und bietet sich daher sehr für ein Refactoring an.

Weiterhin ist die Duplikation in der Pawn Klasse auch sehr anbietend für ein Refactoring. Beim Editieren vom Code ist mir dort auch aufgefallen, dass es sich dort immer für Shotgun Surgery anbietet

Weil der Code in den Verschiedenen Pieces bezüglich des Movements auch sehr viele Duplicates enthält

Programming Principles

Ist es noch dry wenn Pieces eigenen legal moves definieren?

Piece wurde als abstrakte Klasse definiert, damit sich dann auch die verschiedenen Figuren daran orientieren könne und keine eigene Methode für Bewegungen implementieren müssen

Die Piece Klasse folgt dem Liskov Substitution Principle, da sie Kovariant die Beziehung zwischen Der Abstrakten Figur und der konkreten Figur darstellt.

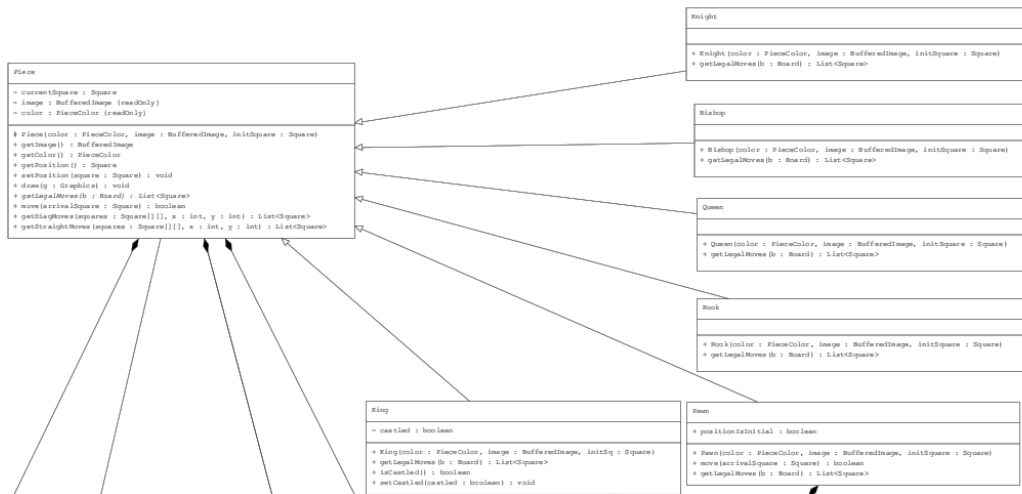
GRASP

Bei der Pure Fabrication Regel kommt die Image Loader und Clock Klasse zum Einsatz. Diese sollen sich natürlich auf wenige Klassen begrenzen. Sie sind vom Rest des Codes isoliert und dienen rein nur zum Verwalten der Uhr und zum Laden der Bilder

Entwurfsmuster

Für die Figuren habe ich mich entschlossen eine Factory zu verwenden. Damit wird auch das Problem der Long Method in der Initialisierungsmethode des Boards verhindert.

Vorher:



Nacher:

Domain Driven Design

Bei Domain Driven Design habe ich viel Wert darauf gelegt, dass die Ubiquitous Language analysiert wurde und sinnvoll umgesetzt wird.

Die Benennung der verschiedenen Klassen wie auch entsprechende Spielfiguren.

Move Methode

Die Entities sind die Klassen die dem Piece Interface zugeordnet sind. Das sind Pawn (Bauer), King (König), Knight (Springer), Rook (Turm), Bishop (Läufer) und Queen (Dame)

Hierbei agiert dann die abstrakte Klasse Piece als Aggregate für alle Figuren.

Als Value Object ist dann beispielsweise die Piece Color Klasse, die die Farbe der Figur angibt. Die Farbe der Figur ist nicht veränderlich.