Programmentwurf Tim-Robin Kalkhof

Inhaltsverzeichnis

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, damit vermieden wird dass die verschiedenen Tests für interferenzen sorgen und unabhängig ausgeführt werden können.

Um die Abhängigkeiten bei der Image Loader Klasse zu reduzieren habe ich Mocks verwendet, damit man hier den Aspekt des Integration Tests vermeiden kann und trotzdem überprüfen kann ob die Funktionalität korrekt ausgeführt wird.

Bei der Code Coverage habe ich besonders den Wert auf die Branch Coverage gelegt, anstatt nur die Coverage Prozentzahl zu priorisieren. Der Grund hierfür war, dass es bei diesem Projekt nicht so viel Sinn ergeben hat auf die Code Coverage zu schauen, da die Zeit limitiert war und die Branch Coverage hier aussagekräftiger ist.

Clean Architecture

Für die Clean Architecture ist die Plugin Schicht das Swift UI und die Test Frameworks AssertJ und Mockito für Mocks, da sie externe Abhängigkeiten sind die nur leicht gekoppelt sind.

Auf der Application Ebene ist sind dann die GUI und Game Klassen vertreten. Diese sorgen für die Verbindung zwischen den Plugins und der Domäne.

Weiterhin sind auf der Domain Ebene die Pieces als Entities vertreten.

Als Abstraction Ebene habe ich den Negamax Algorithmus bestimmt, da dies auf mathematisch fundierten Grundlagen beruht und immer von Grund auf korrekt ist.

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

Refactoring

Die Initiersmethode der Figuren in der Board Klasse ist sehr Lang und Komplex und enthält viele Duplikate und bietet sich daher sehr für ein Refactoring an. Hierbei können viele Aufrufe durch Schleifen verkürzt werden und außerdem wird durhc das bei den Pieces angewandte Factory Pattern

Weiterhin ist die Duplikation in der Pawn Klasse auch sehr anbietend für ein Refactoring. Hierbei wurde dann der Code für beide Farbe in ein If Statement umgebaut. Beim Editieren vom Code ist mir dort auch aufgefallen, dass es dort immer zu Shotgun Surgery kommt.

Weil der Code in den Verschiedenen Pieces bezüglich des Movements auch sehr viele Duplicates enthält, ist es dan nauch sinnvoll diese in die Abstrakte Piece Klasse zu integrieren.

Programming Principles

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

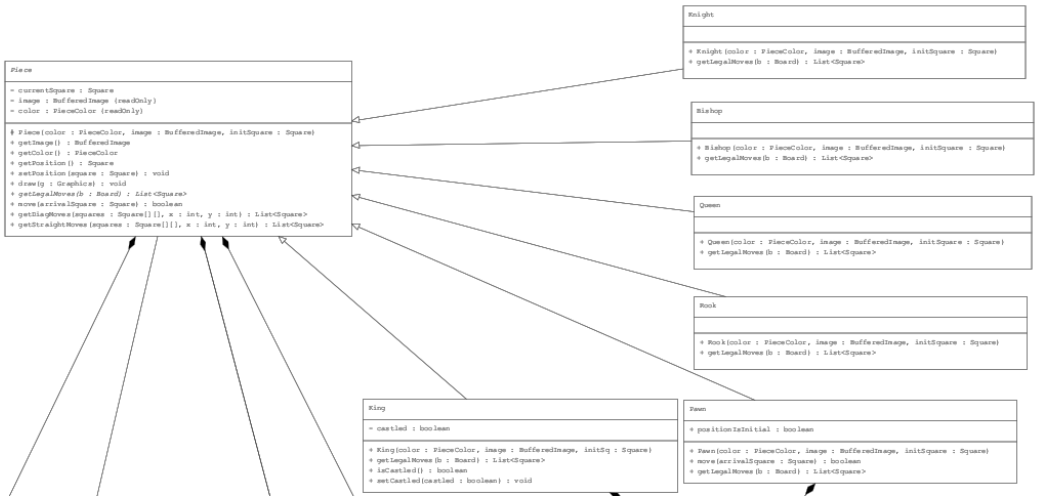
Die Piece Klasse folgt dem Liskov Substitution Principle, da sie Kovariant die Beziehung zwischen der Abstrakten Figur und der konkreten Figur darstellt.

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 Ubiquitos Language analysiert wurde und sinnvoll umgesetzt wird.

Die Benennung der verschiedenen Klassen wie auch entsprechende Spielfiguren, wurde Domainengerecht durchgesetzt.

Dies ist Beispielsweise an der Move Methode zu sehen, die Jede Figur implementiert. Über die sich die Figur (Piece) von einem Feld auf dem Brett (Board) auf ein anderes bewegt.

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.