Programmentwurf für Check-Mate

Moritz Knapp
SICK AG

DAvid Schmidt
SICK AG

March 22, 2022

Contents

1 Intro

Da wir, David und Moritz, schon immer gerne Schach gegeneinander spielten um auch in Lernpausen unseren Geist nicht zu unterfordern, entschieden wir uns ein eigenes Schach zu programmieren. Zudem sind wir davon überzeugt, dass Schach eine perfekte Grundlage für objektorientiertes Programmieren bietet, aufgrund der vielen Figuren die jeweils gerade und schräge Züge individuell ausführen.

2 Clean-Architecture

Bereits vor dem Studium hat David schonmal ein Schach programmiert. Dabei handelte es sich um ca 3000 Zeilen hochineffizienten Arduino Code. In diesem Projrekt wollten wir unseren früheren Ichs beweisen wie viel eleganter so etwas umsetzbar ist. Der erste Schritt dazu ist die Clean-Architecture.

Zusammengefasst ermöglicht eine Clean Architecture die Veränderung der Umgebung, bzw. des Systems, ohne den eigentlichen Kern des Codes anpassen zu müssen. Das bedeutet, dass wir theoretisch unser Spiel um ein echtes Spielbrett mit LEDs und Arduino- Raspberry-Steuerung erweitern könnten, ohne unseren Schach-Code zu verädnern.

2.1 Entitys

Die Entitys sind der Kern unseres Spiels. Unabhängig von der Umgebung sind die einzelnen Figuren virtuell auf einem Feld positioniert und können umplatziert werden. Dadurch ergeben sich grundlegend Folgende Klassen:

- Board
- Square
- Piece

Ein Board besteht aus vielen Squares, auf denen jeweils ein Piece stehen kann. Um dies zu implementieren und später auch verschiedene Pieces zu benutzen, zeigt Figure 1 alle Klassen welche unser Kern der Clean-Architektur beinhalten soll.

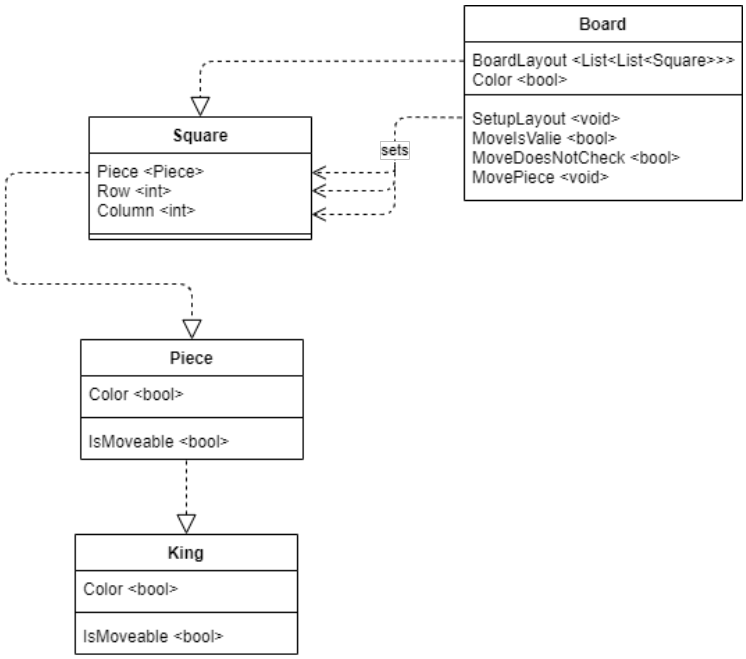

Figure 1: UML-Diagramm der Entitys

Das Diagramm zeigt die Schach-Logik im Kern unserer Clean Architecture, bzw. die Entitys. Das `Board` stellt das Schachbrett dar. Dieses besteht aus 64 `Squares`. Die Methode `SetupLayout` erstellt das Board und besetzt die richtigen Felder mit Pieces mit der entsprechenden Farbe. Von den eigentlichen `Pieces` ist hier nur der König gezeigt, da die anderen Figuren genau wie er die Methode `IsMoveable` implementieren. Das Board bietet drei Methoden, von denen nur eine tatsächlich von weiteren Schichten verwendet wird: `MovePiece`. Die Methode überprüft mit `MoveIsVali` ob es sich um einen Validen Zug handelt. `MoveIsValid` ruft dabei die von Figuren implementierte Methode `IsMoveable` auf um zu überprüfen ob die Figur theoretisch in der Lage wäre den Zug durchzuführen (zB. kann der Läufer schräg dahin?). Ist der Zug möglich wird noch überprüft ob weitere Figuren im Weg stehen und ob der Zug sich selbst in Schach setzen würde (`MoveDoesNotCheck`). Ist alles überprüft, wird der Zug durchgeführt.