# 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

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 folgendes UML-Diagramm alle Klassen welche unser Kern der Clean-Architektur beinhalten soll:

# 3 Programmierprizipien

## 3.1 pros/cons/wieso wir?

## 3.2 SOLID

### 3.2.1 Single Resposibility Principle

### 3.2.2 Open/Closed Principle

### 3.2.3 Liskov' Substitution Principle

### 3.2.4 Interface Segregation Principle

### 3.2.5 Dependency Inversion Principle

Auf Translator anwenden

## 3.3 GRASP

### 3.3.1 Low Coupling

### 3.3.2 High Cohesion

## 3.4 DRY (Don't Repeat Yourself)

Schon umgesetzt?

# 4 Unit Testing

## 4.1 pros/cons/wieso wir?

## 4.2 ATRIP

## 4.3 Code Coverage

## 4.4 Mocking

# 5 Refactoring

## 5.1 Beispiele und Begründungen

## 5.2 Code Smells

# 6 Entwurfsmuster

## 6.1 Diagramme und Begründungen

# 7 Fazit