

3.– 6. September 2012  
in Nürnberg



# Herbstcampus

Wissenstransfer  
par excellence

## Matt in drei Iterationen

Lebendiger Architekturentwurf am Beispiel einer Schach-Engine

Stefan Zörner

oose Innovative Informatik GmbH

# Matt in drei Iterationen

## Lebendiger Entwurf am Beispiel einer Schach-Engine

### **Abstract**

Ein Jahrhunderttraum wie das Fliegen: Eine Maschine, die Menschen im Schach bezwingt. Auch heute für viele Entwickler noch eine faszinierende Aufgabe! Wie zerlegt man das Problem geschickt? Welche wichtigen Entscheidungen sind bei der Umsetzung zu treffen? In diesem Vortrag lernen Sie das Nötigste, um selbst ein Schachprogramm zu bauen. Und Sie erfahren auf vergnügliche Weise ganz nebenbei, wie Sie ganz allgemein eine nachvollziehbare, angemessene Softwarearchitektur entwerfen, bewerten und festhalten können. En passant.

### **Zielgruppe**

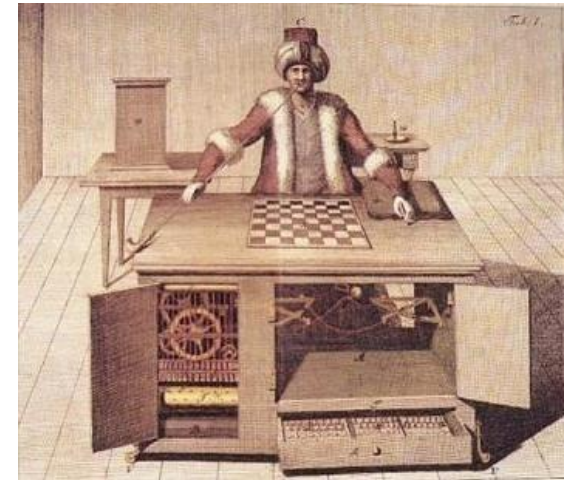
Zielgruppe dieses Vortrags sind in erster Softwareentwickler und -architekten, die neugierig sind, wie eine Schach-Engine funktioniert. Und die anhand dieses Beispiels ein wenig über Architekturentwurf erfahren wollen. Fundierte Schachkenntnisse sind nicht erforderlich.

# Der Schachtürke (Wolfgang von Kempelen, 1769)



*”Meine Damen und Herren, ich habe eine Maschine gebaut wie es sie bisher noch nie gegeben hat: Einen automatischen Schachspieler! Er ist in der Lage, jeden Herausforderer zu schlagen ...”*

(von Kempelen, zu Beginn jeder Vorführung)



## Claude E. Shannon (1916 – 2001)



*“Although perhaps of no practical importance, the question is of theoretical interest, and it is hoped that a satisfactory solution of this problem will act as a wedge in attacking other problems of a similar nature and of greater significance.”*

(aus “Programming a computer to play chess”, 1949)

- amerikanischer Mathematiker, Kryptologe, ...
- Begründer der Informationstheorie
- Bahnbrechend für Computerschach: „Programming a computer to play chess“

## Stefan Zörner ....

... bei oose seit 2006 als Trainer und Berater

... regelmäßig Trainings und Workshops zu:

- Softwareentwurf und -architektur, insbesondere Architekturdokumentation
- Umsetzung mit Java-Technologien

... war auf der Suche nach einem lebendigen Beispiel für

- Entwurfsprinzipien und –muster
- Architekturentwurf und vor allem: Architekturdokumentation



- Fasziniert vom Thema
- neugierig, wie aufwendig eine eigene Umsetzung tatsächlich wäre

... selbstmäßiger Gelegenheitsschachspieler



## Mission Statement – 2 Ziele für den Vortrag

# 1

Ihr erfahrt die Antwort auf die Frage:  
Wie schreibe ich eine eigene Schach-Engine?

- Ihr seid am Ende mit dem Wissen ausgestattet, selbst eine zu schreiben,
- bzw. Ihr könnt abschätzen, wie aufwendig das wäre.

(Spaß-Teil)

# 2

Ihr erfahrt nebenbei etwas über  
Architekturentwurf, -bewertung, -dokumentation

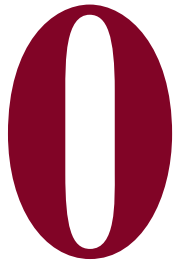
- Wir hacken nicht einfach drauf los, sondern gehen methodisch vor.

(Ernst-Teil)

# Agenda

- 0** Die Aufgabe
- 1** Iteration 1: „Der Durchstich“
- 2** Iteration 2: „Das Bauerndiplom“
- 3** Iteration 3: „Der Taktikfuchs“
- 4** Ausblick und Weitere Informationen

# Agenda

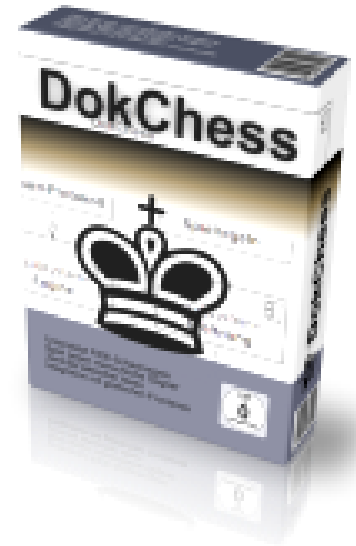


- 0** Die Aufgabe
- 1 Iteration 1: „Der Durchstich“
- 2 Iteration 2: „Das Bauerndiplom“
- 3 Iteration 3: „Der Taktikfuchs“
- 4 Ausblick und Weitere Informationen



## „DokChess“ – Ziele und Features

- DokChess ist eine voll funktionsfähige Schachengine
- Sie dient als einfach zugängliches und zugleich ungemein attraktives Fallbeispiel für Architekturentwurf, -bewertung und -dokumentation.
- Der verständliche Aufbau lädt zum Experimentieren und zum Erweitern der Engine ein
- Ziel ist nicht die höchstmögliche Spielstärke – dennoch gelingen Partien, die Gelegenheitsspielern Freude bereiten.



## Wesentliche Features

- Vollständige Implementierung der FIDE-Schachregeln
- Unterstützt das Spiel gegen menschliche Gegner und andere Schachengines
- Beherrschung zentraler taktischer Ideen, beispielsweise Gabel und Spieß
- Integration mit modernen graphischen Schach-Frontends

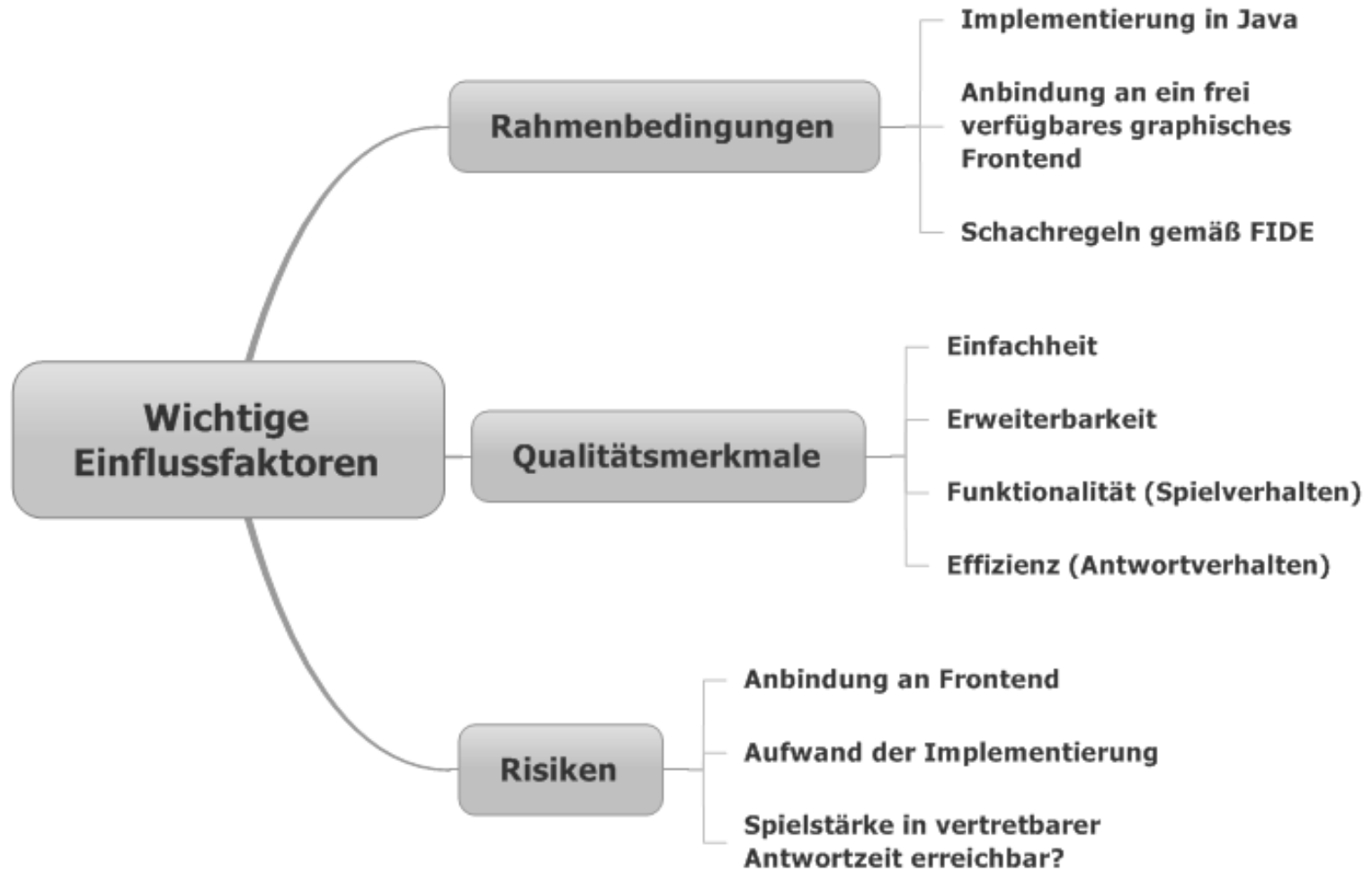
# Softwarearchitektur. Eine (!) Definition



*“Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.”* (Eoin Woods)

- Architekturentscheidungen sind diejenigen, die sich im weiteren Verlauf nur sehr schwer revidieren lassen.
- Konsequenzen: höhere Kosten, Zeitverlust, ggf. scheitert das Vorhaben

# Einflussfaktoren auf Entscheidungen



## Jetzt: Drei Iterationen

1

2

3

### Gleichförmiger Aufbau in der Darstellung

- Zu Beginn: Vorstellung des Iterationszieles
- Darstellung zentraler Konzepte, Entscheidungen
- Tipps und Tricks
- Am Ende: Spiel gegen die Engine, Bewertung

# Agenda

# 1

- 0** Die Aufgabe
- 1** Iteration 1: „Der Durchstich“
- 2** Iteration 2: „Das Bauerndiplom“
- 3** Iteration 3: „Der Taktikfuchs“
- 4** Ausblick und Weitere Informationen

# 1. Iteration („Durchstich“), Steckbrief



## Ziel:

Engine interagiert mit menschlichem Spieler über ein graphisches Frontend. Es entwickelt sich eine “Partie” über mehrere Züge.

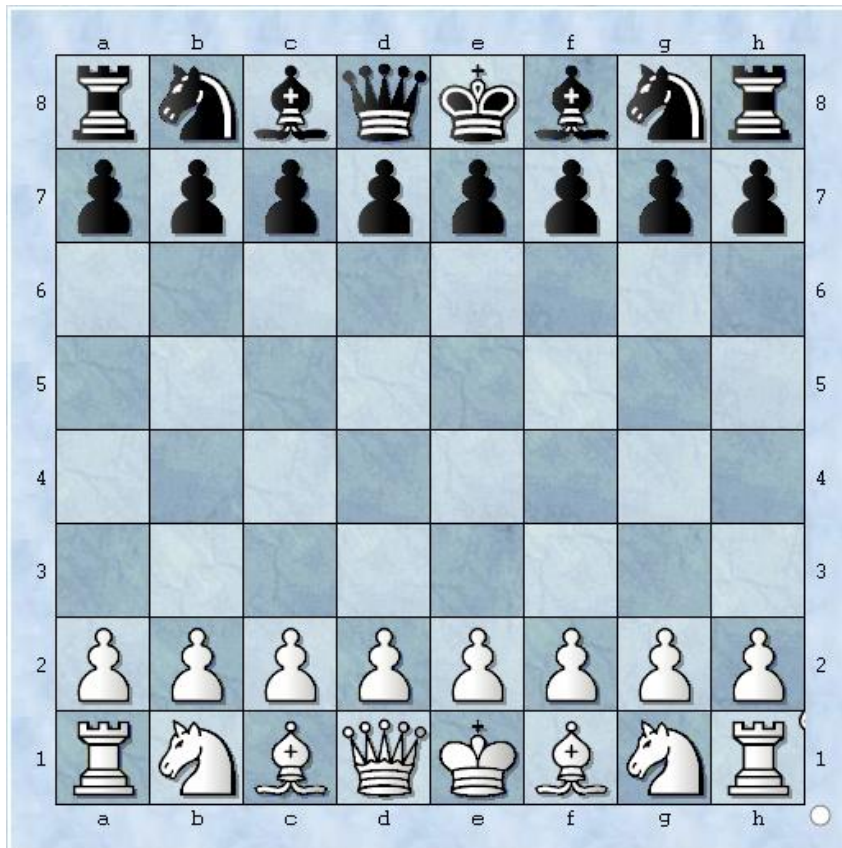
## Zentrale Entscheidungen:

- Darstellung der Spielsituation („Stellung“)
- Auswahl eines graphischen Frontends

## Implementierungsaufgaben

- Darstellung des „Brettes“, Felder, Züge, etc.
- Anbindung an das graphische Frontend
- Trivialer Zuggenerator

# Die Schachdomäne



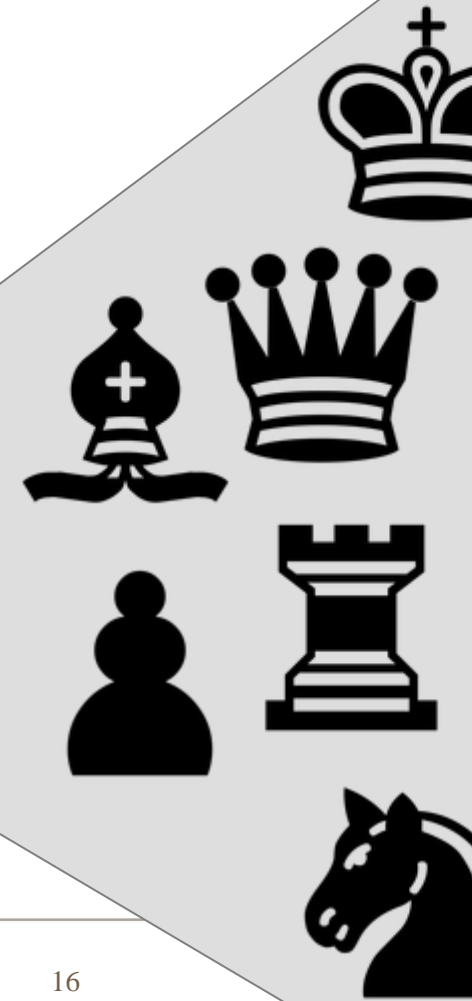
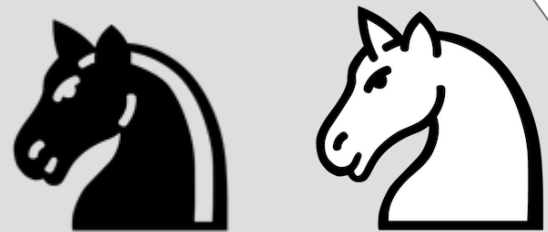
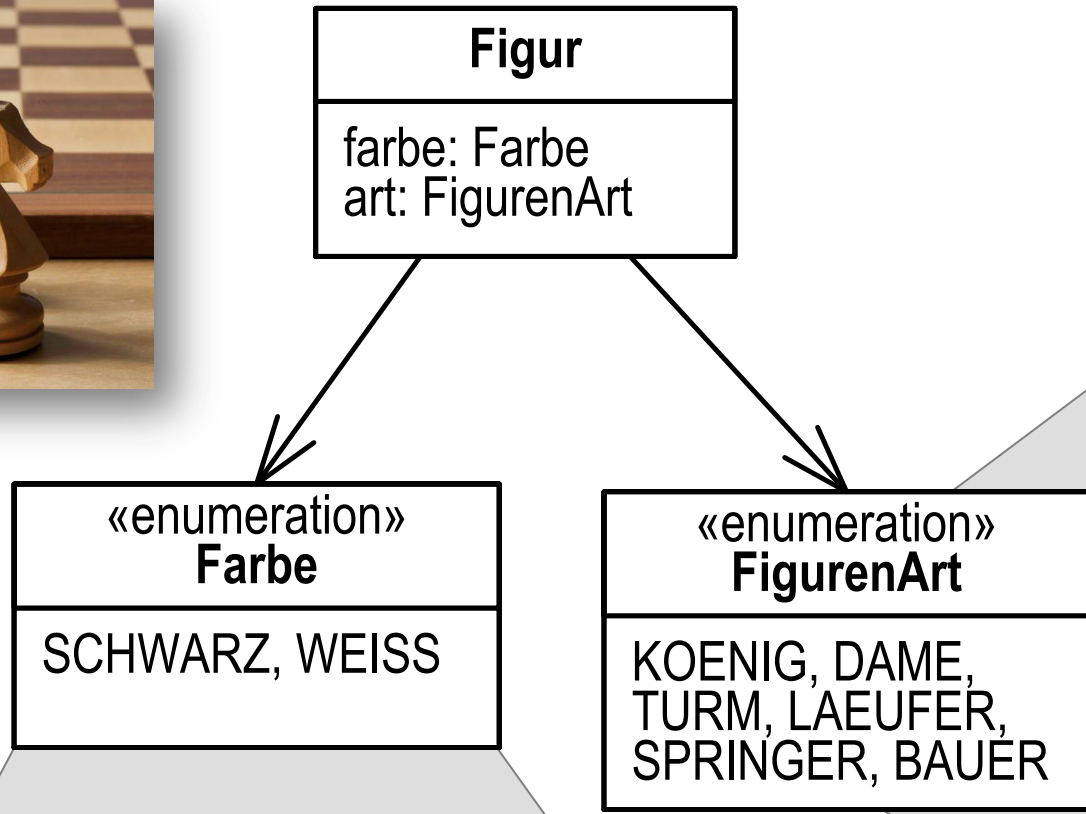
„Das Schachspiel wird zwischen zwei Gegnern gespielt, die abwechselnd ihre Figuren auf einem quadratischen Spielbrett, Schachbrett genannt, ziehen.“

FIDE-Regeln

- Schachbrett 8 x 8 Felder
- 8 Reihen (1-8) und 8 Linien (a-h)
- 2 Spieler, Farben: Schwarz und Weiß
- Figurenarten: König, Dame, Turm, Läufer, Springer, Bauer

- Gezogen wird von Feld zu Feld, gegnerische Figuren werden „geschlagen“
- Ziel: den gegnerischen König zu fangen („Schach matt“)

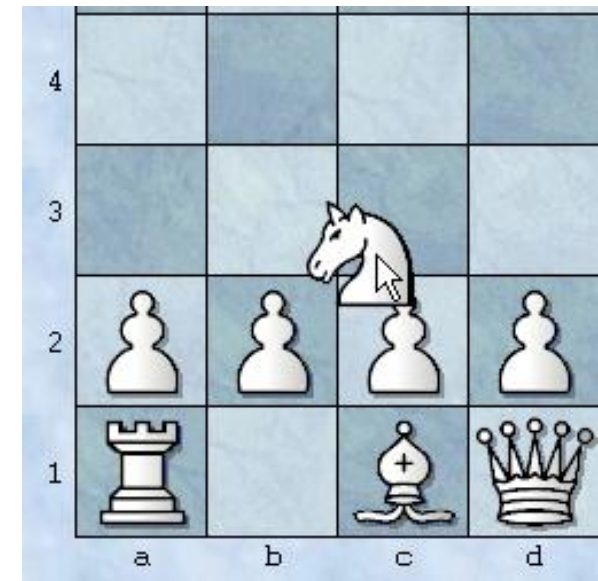
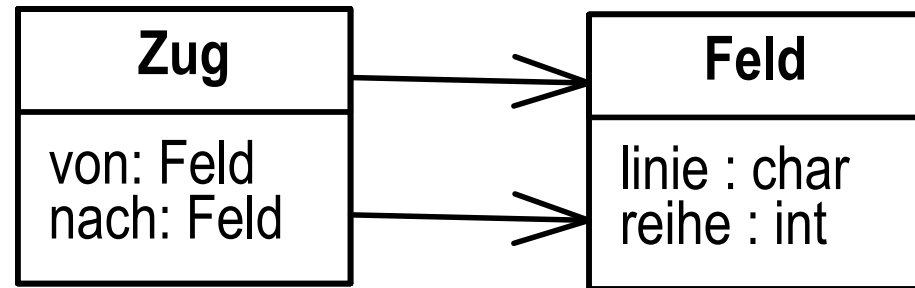
# Einfache Darstellung von Figuren



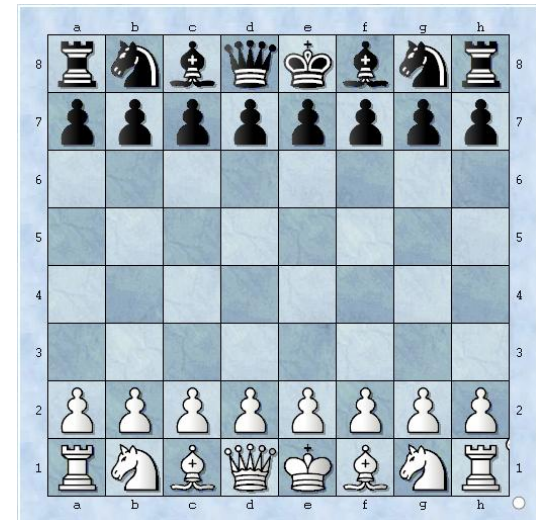
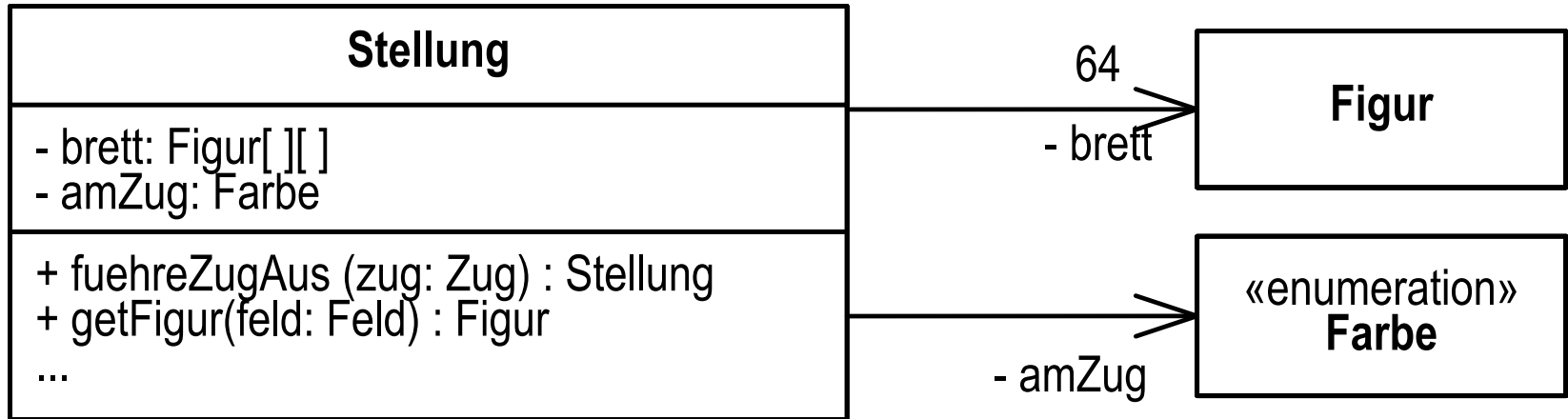


# Züge und Felder als Klassen

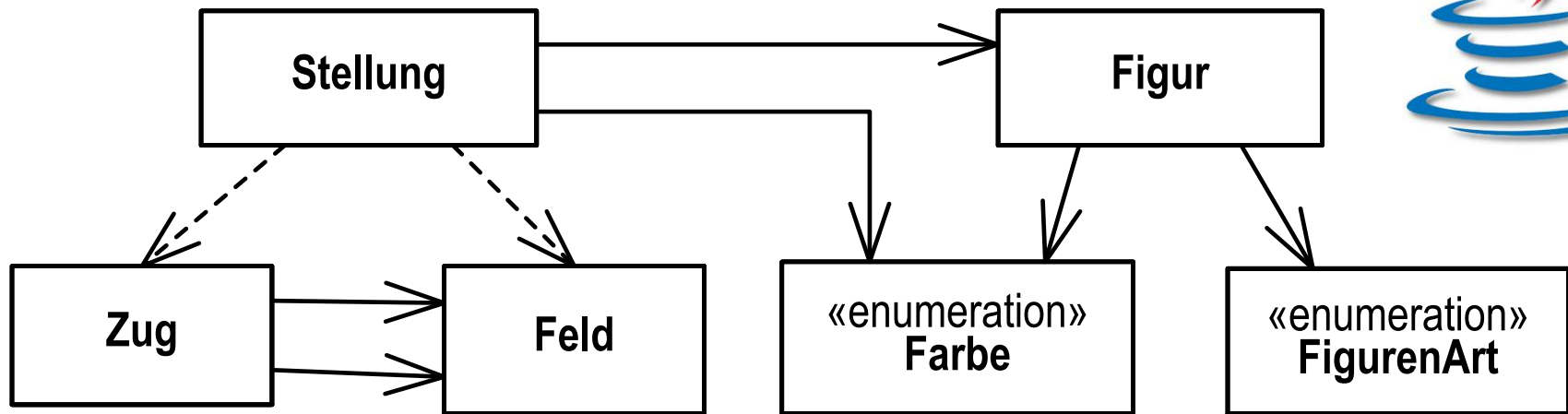
b1 – c3



# Einfache Darstellung der Stellung



# Implementierung in Java



- 4 Klassen
- 2 Aufzählungstypen
- Unit-Tests für Stellung

```

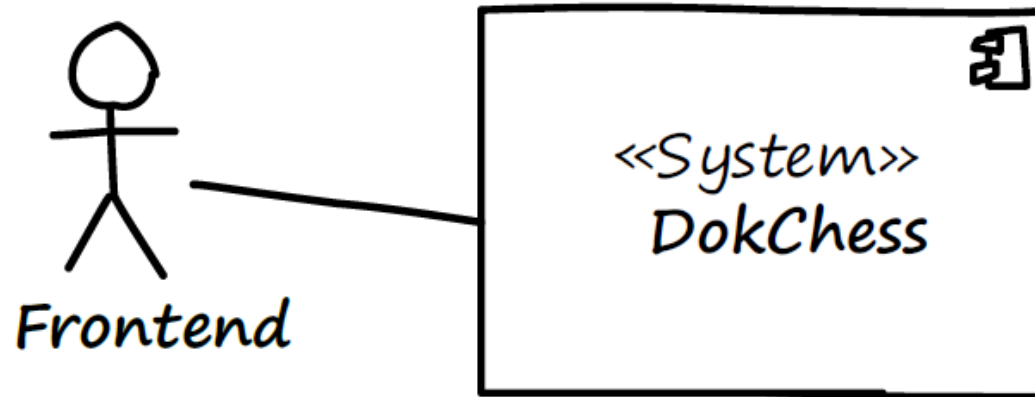
Stellung.java
/**
 * Beschreibt eine Spielsituation. Hierzu zaehlen die Figuren auf dem
 * Brett(Stellung), die Farbe am Zug, Rochaderechte usw.
 *
 * @author StefanZ
 */
public class Stellung {

    private Farbe amZug;

    private Figur[][] brett;

    public Stellung() {
        this.amZug = Farbe.WEISS;
        this.brett = new Figur[8][8];
    }
  
```

# Auswahl + Anbindung graphisches Frontend



## Zentrale Anforderungen an Frontend:

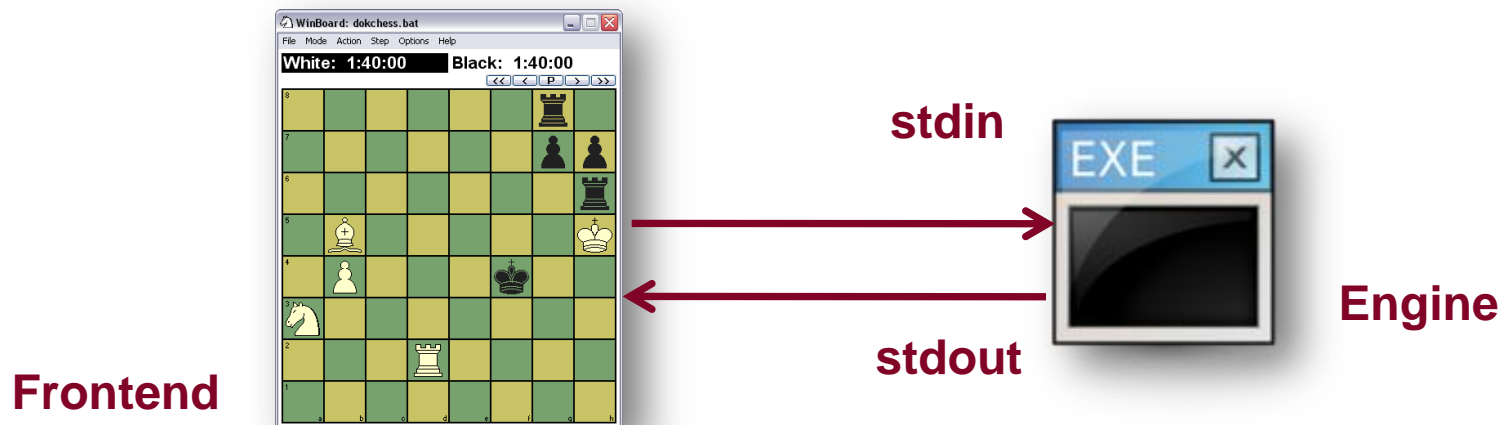
- Auf Windows lauffähig, idealerweise kostenfrei
- Anbindung einer eigenen Engine über ein geeignetes Protokoll möglich

## Recherche ergibt:

- Mehrere Lösungen für verschiedene Betriebssysteme verfügbar
- sowohl frei als auch kommerziell
- 2 etablierte Protokolle

# Protokolle für Schach-Engines/-Frontends

- 2 Lösungen etabliert/dokumentiert
- beide ASCII-basiert, beide via stdin/stdout



Universal Chess Interface (UCI)

<http://wbec-ridderkerk.nl/html/UCIProtocol.html>

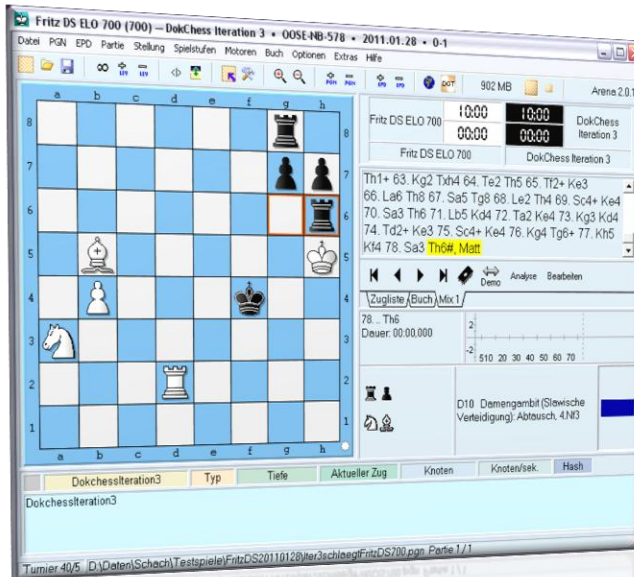
Chess Engine Communication Protocol („Xboard/WinBoard“)

<http://home.hccnet.nl/h.g.muller/engine-intf.html>

# Betrachtete Schach-Frontends

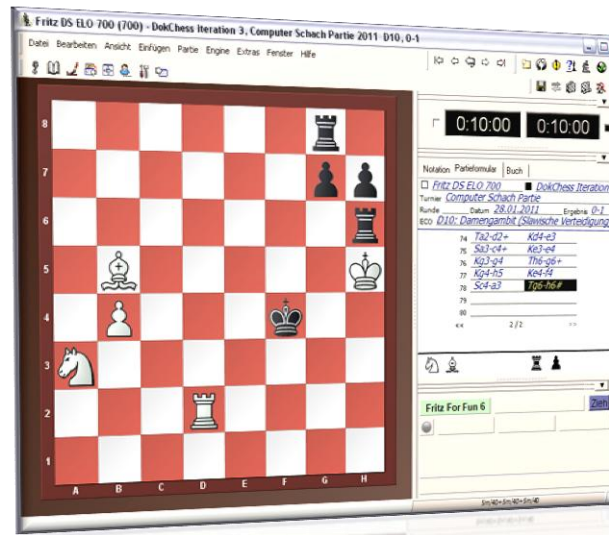
## Arena 3.0

<http://www.playwitharena.com>



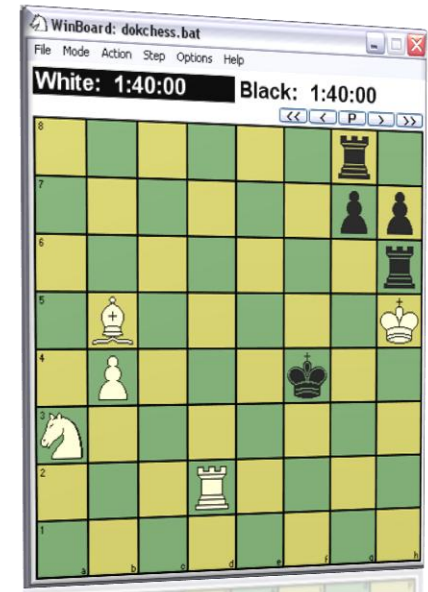
## Fritz for Fun 6

<http://www.chessbase.de/>



## WinBoard 4.4.4

<http://tim-mann.org/xboard.html>





# Demo: Eine Partie gegen DokChess::Iteration1



# Agenda

## 2

- 0 Die Aufgabe
- 1 Iteration 1: „Der Durchstich“
- 2 Iteration 2: „Das Bauerndiplom“
- 3 Iteration 3: „Der Taktikfuchs“
- 4 Ausblick und Weitere Informationen



## 2. Iteration („Bauerndiplom“), Steckbrief



### Ziel:

Die Engine spielt Partien korrekt.

### Zentrale Entscheidungen:

- Grundlegende Zerlegung in Subsysteme (Grundriss)
- Festlegung von Abhängigkeiten zwischen diesen

### Implementierungsaufgaben

- Spielregeln

## Zum Namen „Bauerndiplom“

*„Das Bauerndiplom bescheinigt einem Spieler, dass er die Grundregeln des Schachs beherrscht. ... Die Prüfung des Deutschen Schachbundes gibt jedem die Möglichkeit, sein Schachwissen erfolgreich zu testen.“*

aus „Schach Zug um Zug“

### Idee:

Am Ende der Iteration absolviert die Engine die insgesamt 8 Aufgaben mit Hilfe eines automatisierten Tests.

### Schach Zug um Zug

Bauerndiplom, Turmdiplom, Königsdiplom

Offizielles Lehrbuch des Deutschen Schachbundes zur Erringung der  
Diplome

Autor: Helmut Pfleger

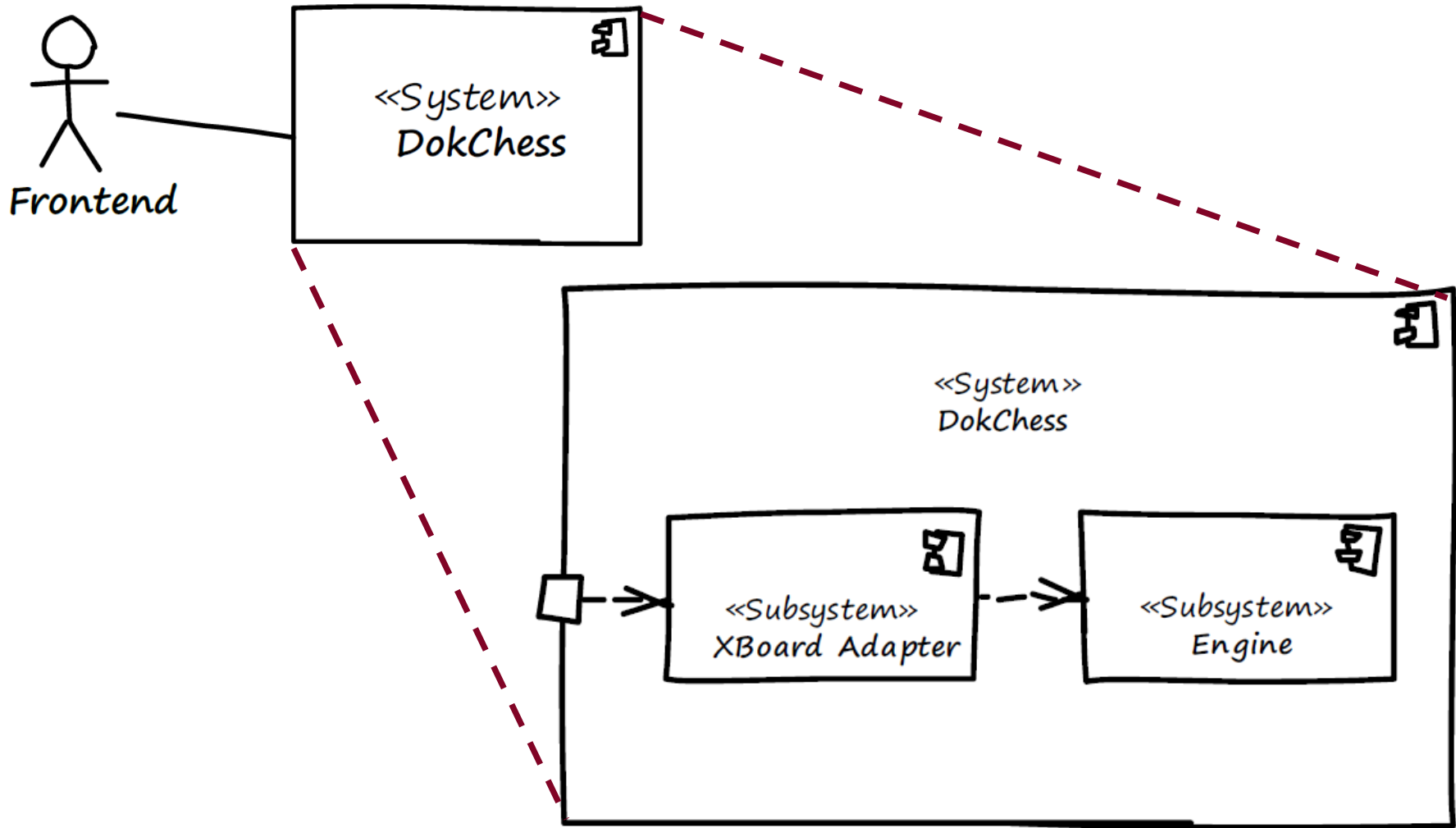
Gebundene Ausgabe: 272 Seiten

Bassermann Verlag; 5. Auflage Januar 2004

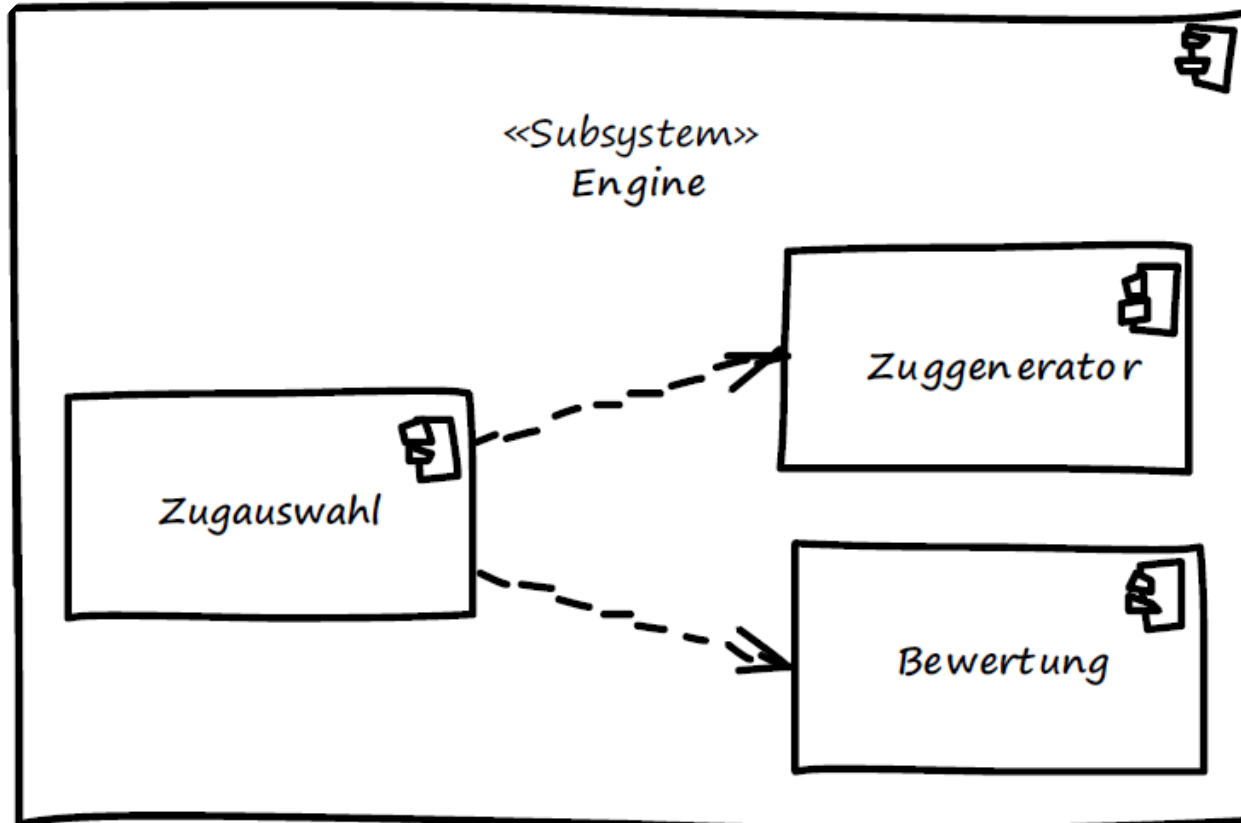
ISBN-13: 978-3809416432



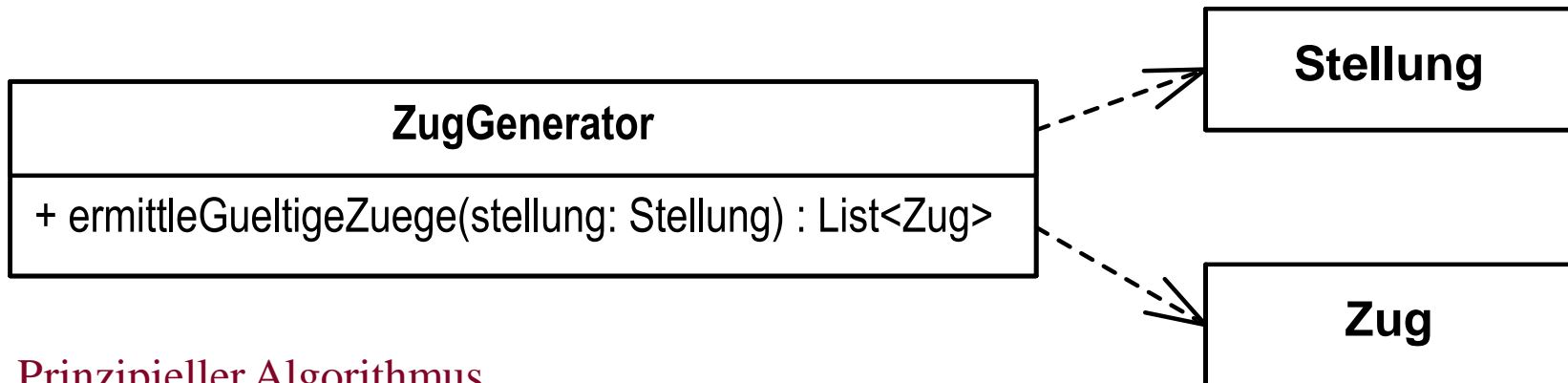
# Bausteinsicht



## Bausteinsicht Engine-Subsystem, Ebene 2



## Zentrale Implementierungsaufgabe: Der Zuggenerator



### Prinzipieller Algorithmus

- Ermittle Farbe am Zug (aus Stellung)
- Prüfe für jede Figur der Farbe, wo sie überall hinziehen kann (freie Felder, ggf. schlagen einer gegnerischen Figur)
- Füge jeweils einen Zug in die Ergebnisliste

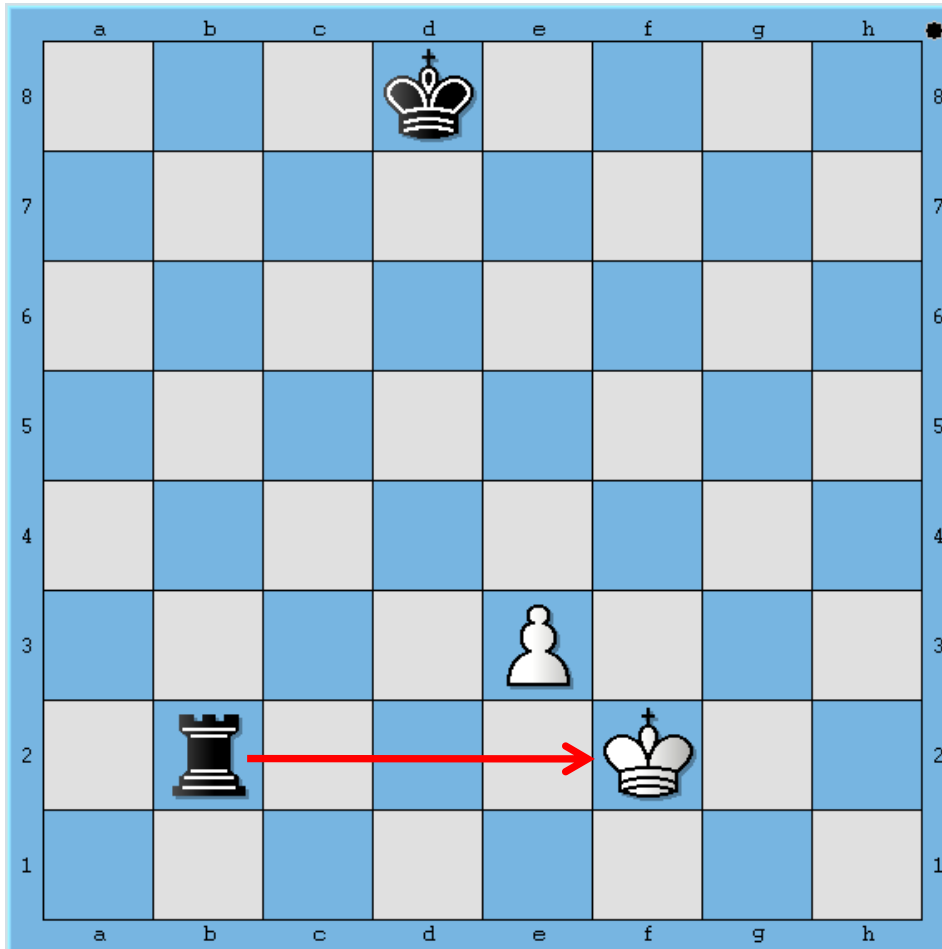
### Implementierung

Im Grunde einfach, aber sehr aufwändig (bei mir: 10 Klassen 580 TLOC)

(6 verschiedene Figurenarten, Rochade, en passant)

Überprüfung auf gültige Züge schwierig wg. Schachgebot des Gegners

## Problem Schachgebot (Einfaches Beispiel: Fesselung)



Weiß am Zug.

Lokal betrachtet:

e2-e3 und e2-e4 sehen gut aus  
(Zielfelder frei)

Nach Ausführung e2-e3:

Weiss im Schach

Züge sind beide ungültig!

Dürfen nicht in Ergebnisliste!

# Einfache (naive?) Lösung

## Prinzipieller Algorithmus

- Ignoriere Problem mit Schachgebot zunächst
  - ➔ Liste von Zugkandidaten („pseudolegaler“ Züge)
- Für jeden Zugkandidaten:
  - Führe Zug aus
  - Prüfe neue Stellung auf Schachgebot.
    - Ja: ➔ Zug kann verworfen werden (ungültig)
    - Nein ➔ Zug kann in Ergebnisliste

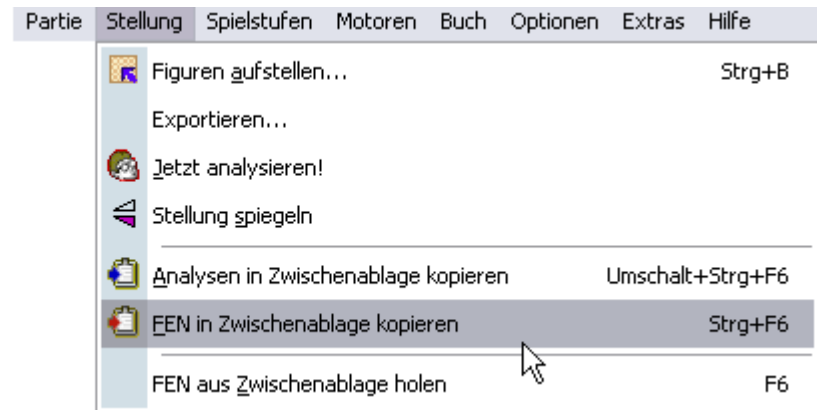
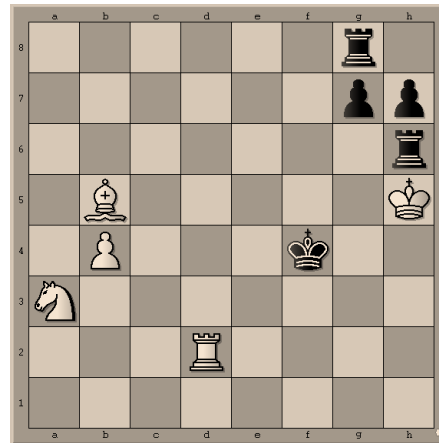
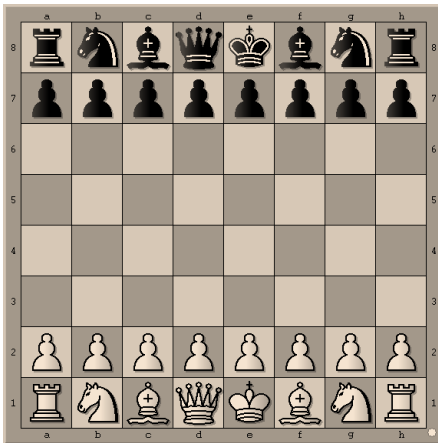


# Praxistipp : Forsyth-Edwards-Notation

„Die Forsyth-Edwards-Notation (FEN) ... ist eine Kurznotation, mit der jede beliebige Brettstellung im Schach niedergeschrieben werden kann.“

wikipedia.de

## 2 Beispiele:

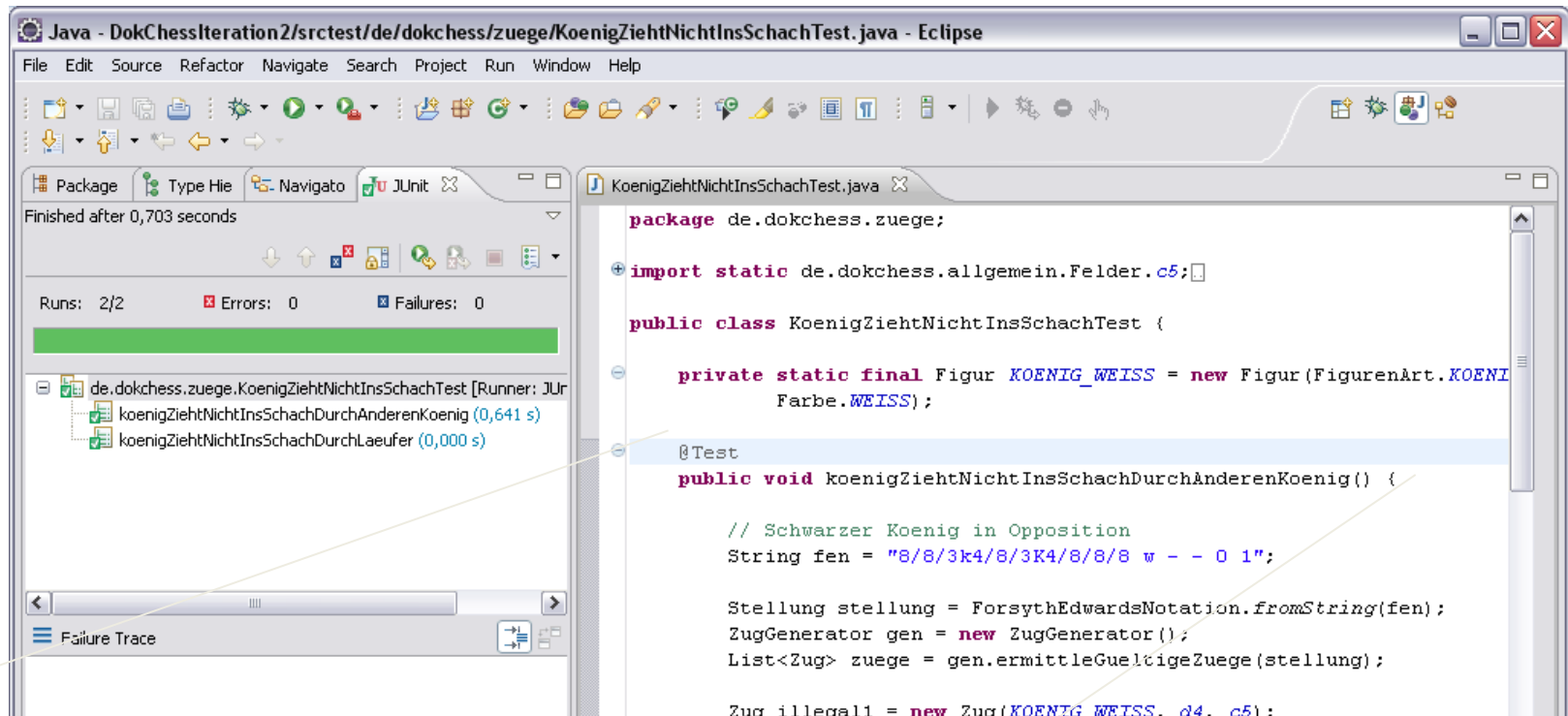


1. "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1"

2. "6r1/6pp/7r/1B5K/1P3k2/N7/3R4/8 w - - 30 79"



# Einsatz von FEN in Unit-Tests






```

@Test
public void koenigZiehtNichtInsSchachDurchAnderenKoenig() {

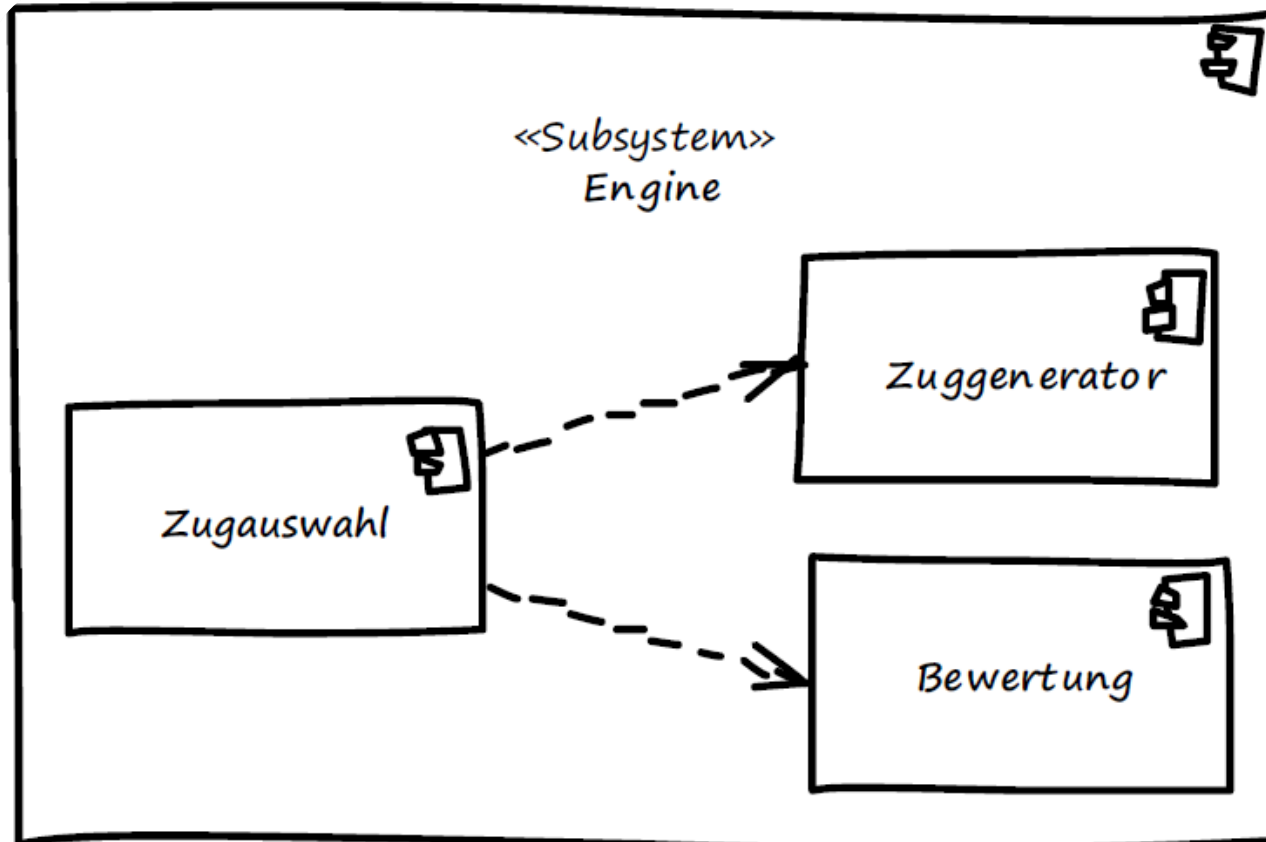
    // Schwarzer Koenig in Opposition
    String fen = "8/8/3k4/8/3K4/8/8/8 w - - 0 1";
    
```

```

        assertTrue("Zug illegal1", !gen.isLegal(zuege, illegal1));
        assertTrue("Zug illegal2", !gen.isLegal(zuege, illegal2));
        assertTrue("Zug illegal3", !gen.isLegal(zuege, illegal3));
    }
}
    
```

 KOENIG_WEISS : Figur
 koenigZiehtNichtInsSchachDurchAnderenKoenig() : void
 koenigZiehtNichtInsSchachDurchLaeufer() : void

# Bausteinsicht Engine-Subsystem, Ebene 2



## Naive Implementierungen für Bewertung und Auswahl

### Bewertung einer Stellung anhand des Materials

- Jede Figur erhält einen Wert (z.B. Bauer 1, ... Dame 9)
- Eigene Figuren zählen positiv, gegnerische negativ
- Werte aufsummieren, je höher der Wert, je besser die Stellung



1



3



3



5



9

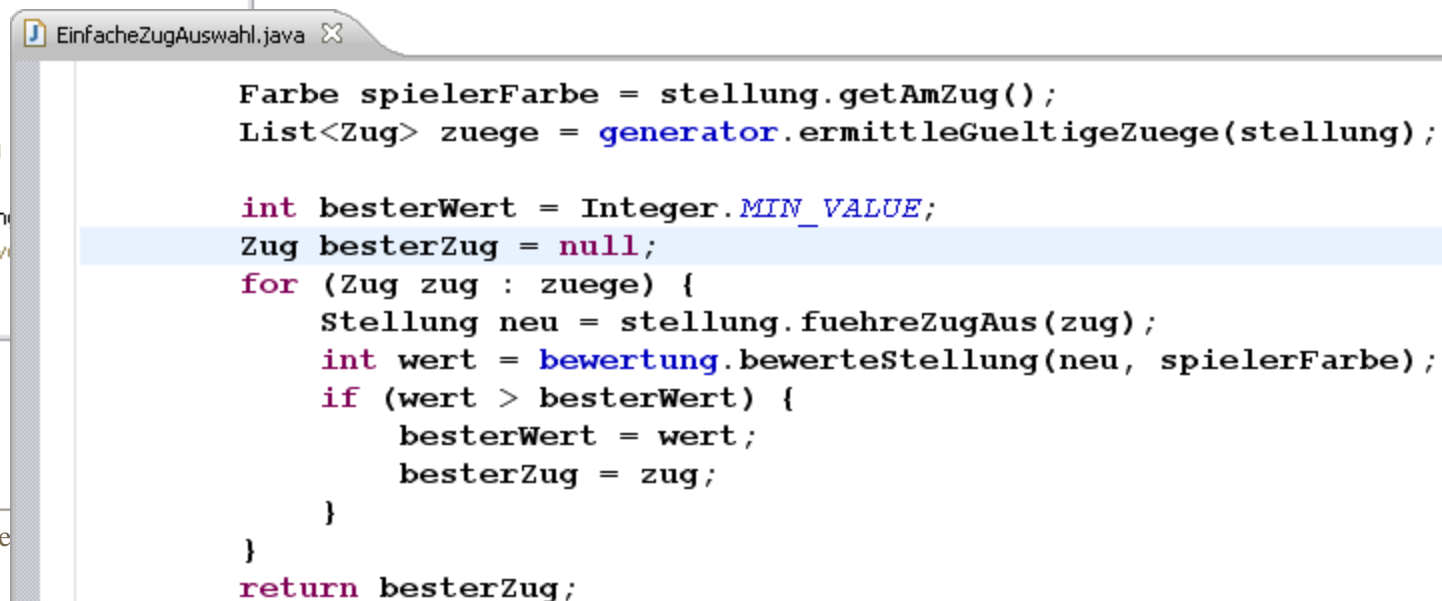
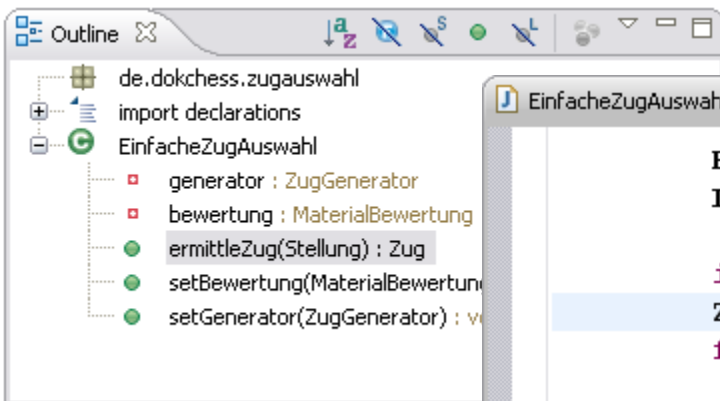
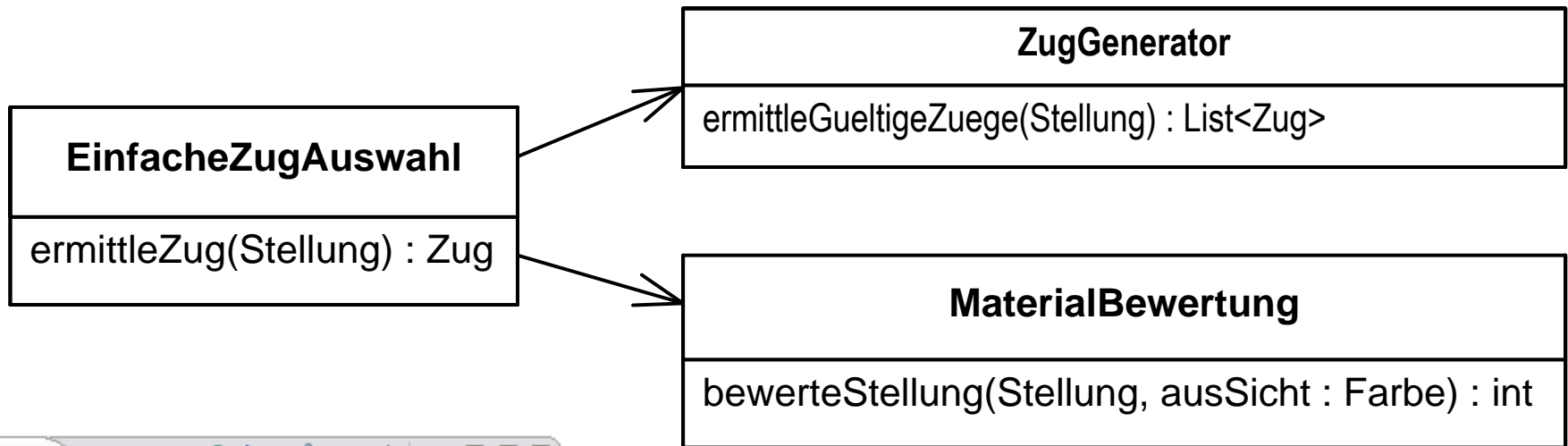


-

### Einfache Auswahl aus der Liste der gültigen Züge:

- Jeden Zug auf die aktuelle Stellung anwenden
- Resultierende Stellung bewerten (s.o.)
- Zug mit bestem Ergebnis wird ausgewählt

# Klassendiagramm der Engine, Implementierung



```
EinfacheZugAuswahl.java

Farbe spielerFarbe = stellung.getAmZug();
List<Zug> zuege = generator.ermittleGueltigeZuege(stellung);

int besterWert = Integer.MIN_VALUE;
Zug besterZug = null;
for (Zug zug : zuege) {
    Stellung neu = stellung.fuehreZugAus(zug);
    int wert = bewertung.bewerteStellung(neu, spielerFarbe);
    if (wert > besterWert) {
        besterWert = wert;
        besterZug = zug;
    }
}
return besterZug;
```

# Demo: Eine Partie gegen DokChess::Iteration2

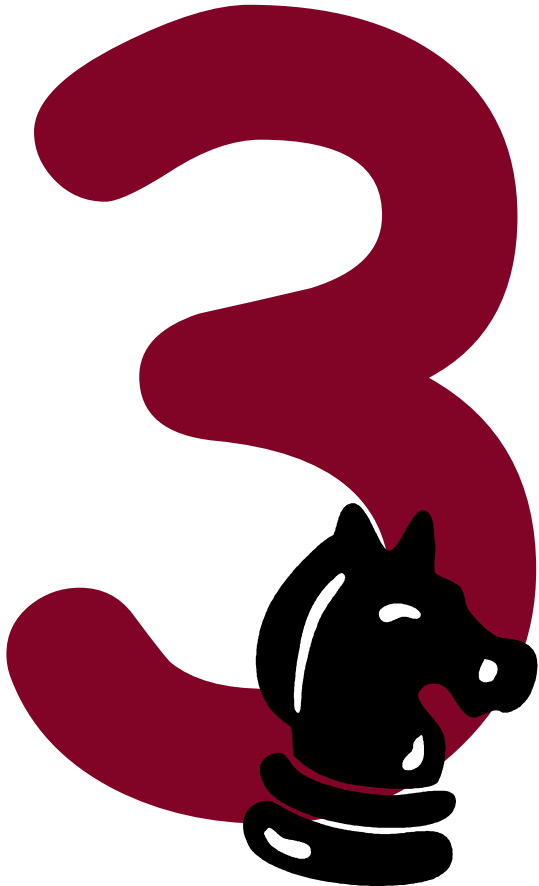


# Agenda

## 3

- 0 Die Aufgabe
- 1 Iteration 1: „Der Durchstich“
- 2 Iteration 2: „Das Bauerndiplom“
- 3 Iteration 3: „Der Taktikfuchs“
- 4 **Ausblick und Weitere Informationen**

### 3. Iteration („Taktikfuchs“), Steckbrief



#### Ziel:

Die Engine spielt sinnvolle Partien.

#### Zentrale Entscheidungen:

- Auswahl der Algorithmen für Stellungsbewertung und Zugauswahl (Architekturentscheidungen?)

#### Implementierungsaufgaben

- Zugauswahl, Bewertung

## Zum Namen „Taktikfuchs“

*Die Taktik ist der Teil, bei dem es direkt zur Sache geht, wenn der Spieler in Gedanken spricht: „Wenn ich da hin ziehe und er dort hin ... dann schlage ich seinen Bauern ... was kann er nun als nächstes unternehmen ... usw.“*

aus „Schachtaktik: Wie ich ein Taktikfuchs werde“

### Idee:

Am Ende der Iteration erkennt und spielt bzw. verhindert die Engine einfache taktische Motive, wenn sie sich ergeben.

**Schachtaktik**  
**Wie ich ein Taktikfuchs werde**  
ab 8 Jahren  
Garri Kasparow  
Broschiert: 98 Seiten  
Verlag: Edition Olms 2010  
Sprache: Deutsch  
ISBN-13: 978-3283010157





# Minimax-Algorithmus – Ein bisschen Theorie

*”Der Minimax-Algorithmus ist ein Algorithmus zur Ermittlung der optimalen Spielstrategie für bestimmte Spiele, bei denen zwei gegnerische Spieler abwechselnd Züge ausführen. Die Minimax-Strategie sichert ... höchstmöglichen Gewinn bei optimaler Spielweise des Gegners.”*

<http://de.wikipedia.org/wiki/Minimax-Algorithmus>

## Die Natur des Schachspiels

- nicht vom Zufall abhängig
- offen, d. h. in jeder Spielsituation sind jedem der beiden Spieler alle Zugmöglichkeiten des jeweiligen Gegenspielers bekannt



# Minimax – Bereits in Zugauswahl in Iteration 2

## Schritte bisher

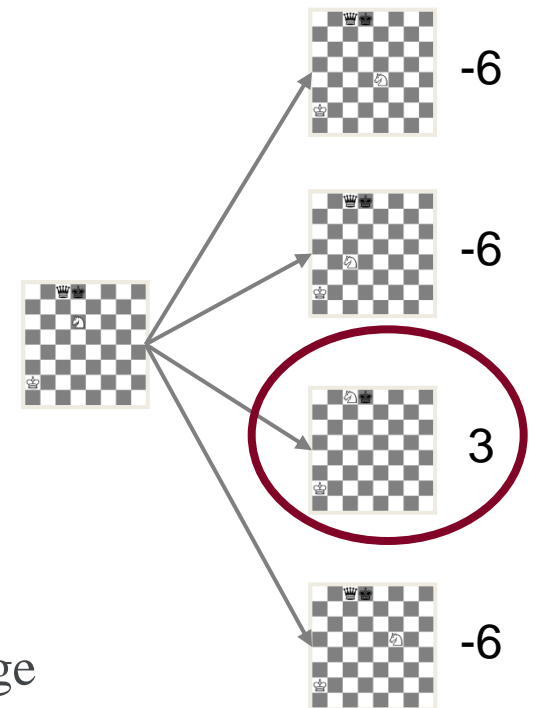
- Ermittle aus aktueller Stellung alle mögliche Züge
- Mache jeweils Zug, und bewerte neue Stellung
- Wähle das Maximum aus

## Problem

- Reaktion des Gegners wird ignoriert
- Und „meine“ Reaktion auf diese Reaktion, usw.

## Lösungsidee

- Ermittle Suchbaum mit eigenen und gegnerischen Züge
- Bewerte Blätter (Baum bis zu bestimmter Tiefe)
- Finde jeweils den für mich/den Gegner besten Zug.



# Minimax, 2 Halbzüge

## Berechnung des Spielbaums

- fixe Tiefe, hier 2

## Bewertung der „Terminalknoten“

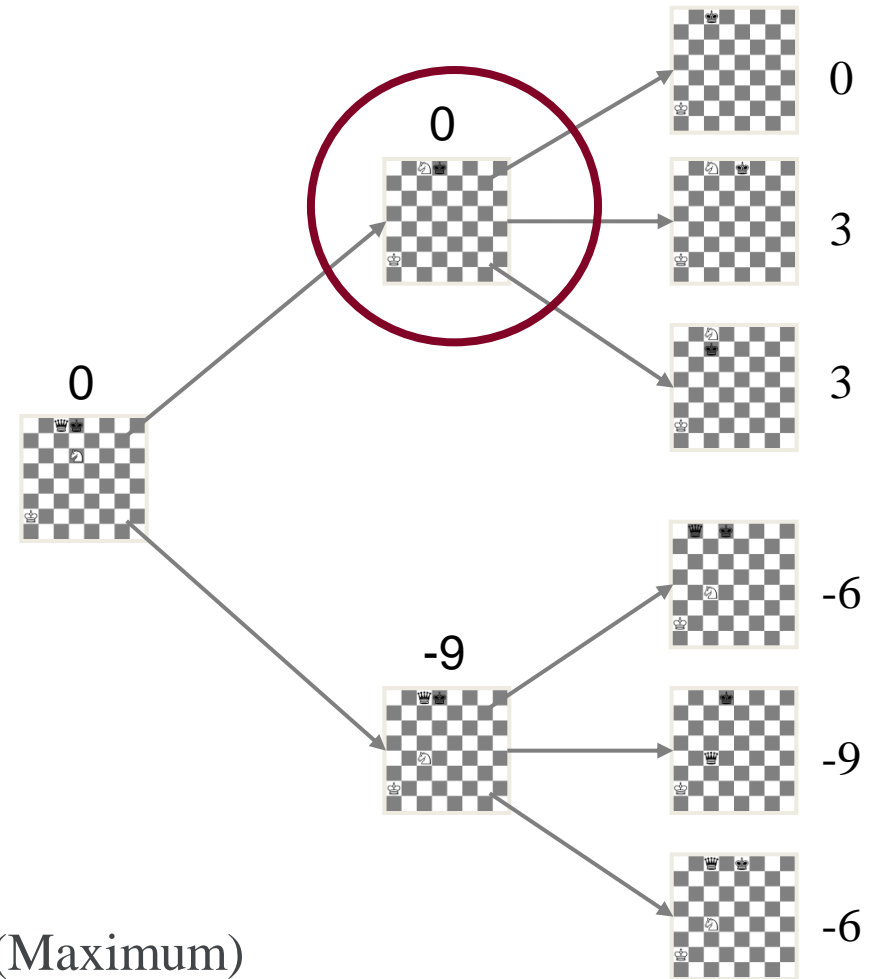
- Anwendung der Bewertungsfunktion aus „meiner“ Sicht

## Minimum für den Gegner

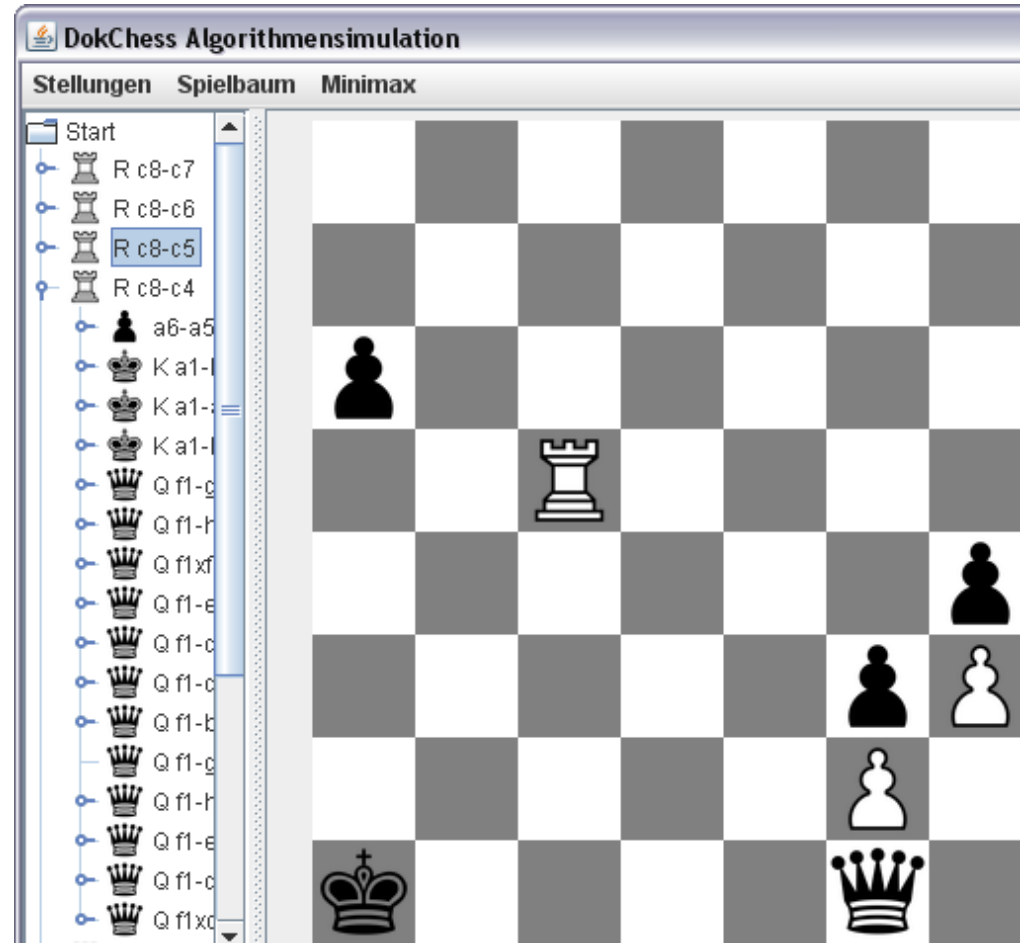
- Was ist jeweils der beste Zug für den Gegner (Minimum)

## Maximum für mich

- Was ist jeweils der beste Zug für mich (Maximum)




# Demo Spielbaum + Minimax



# Matt und Patt

## Ein letztes, aber gravierendes Problem

- Minimax mit fester Tiefe und rein materialbasierte Bewertung kennt weder Matt noch Patt
- Konsequenz: Das Programm gewinnt Material, aber keine vermutlich Partie



*„Das Ziel eines jeden Spielers ist es, den gegnerischen König so anzugreifen, dass der Gegner keinen regelgemäßen Zug zur Verfügung hat. Der Spieler, der dieses Ziel erreicht, hat den gegnerischen König mattgesetzt und das Spiel gewonnen.“*

*„Die Partie ist remis (unentschieden), wenn der Spieler, der am Zuge ist, keinen regelgemäßen Zug zur Verfügung hat und sein König nicht im Schach steht. Eine solche Stellung heißt Pattstellung.“*

FIDE-Schachregeln

# Eine Lösung für Matt und Patt

## Lösung im Minimax-Algorithmus

- Falls der Zuggenerator beim Explorieren keine gültigen Züge für eine Stelle findet:

Prüfe, ob die Seite am Zug im Schach steht

**Ja** → Matt, bewerte Knoten maximal bzw. minimal (je nach Sicht, wenn ich selbst Matt bin minimal)

**Nein** → Bewerte den Knoten ausgeglichen (Wert: 0)

So wird ein Matt einem Materialgewinn vorgezogen, das eigene Matt wenn möglich verhindert, Patzer vermieden, ...

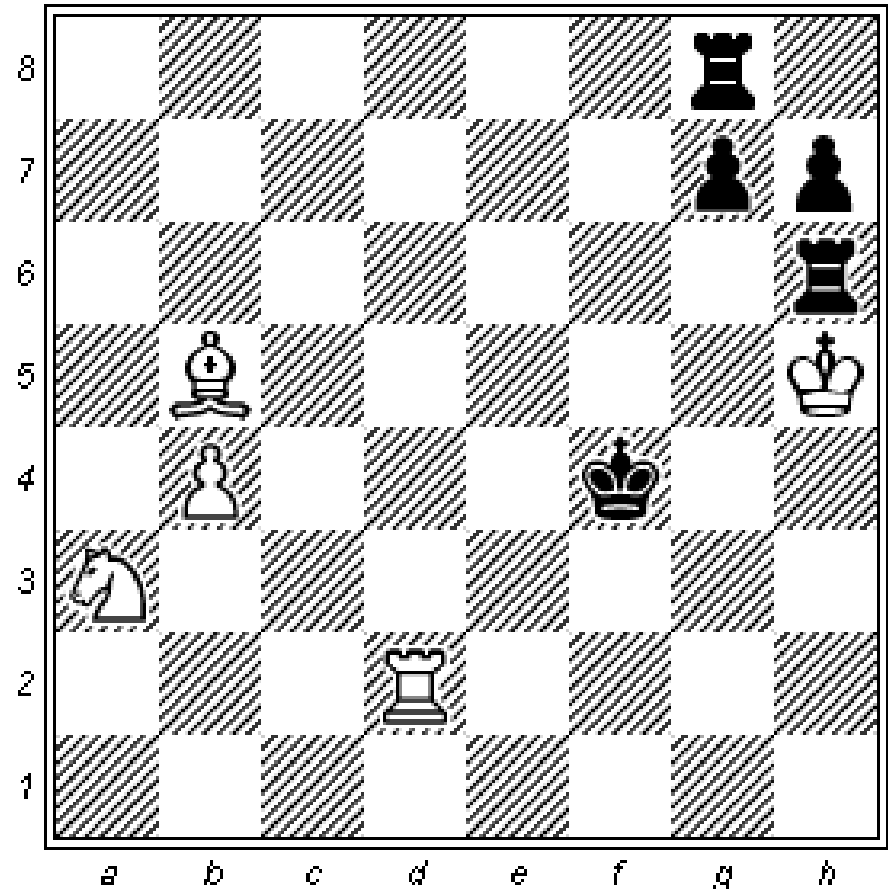


# Demo: Eine Partie gegen DokChess::Iteration3



# DokChess gegen Fritz DS

Fritz DS - DokChess Iter 3  
Buchholz 2011



Schwarz setzt nach 78 Zügen matt.

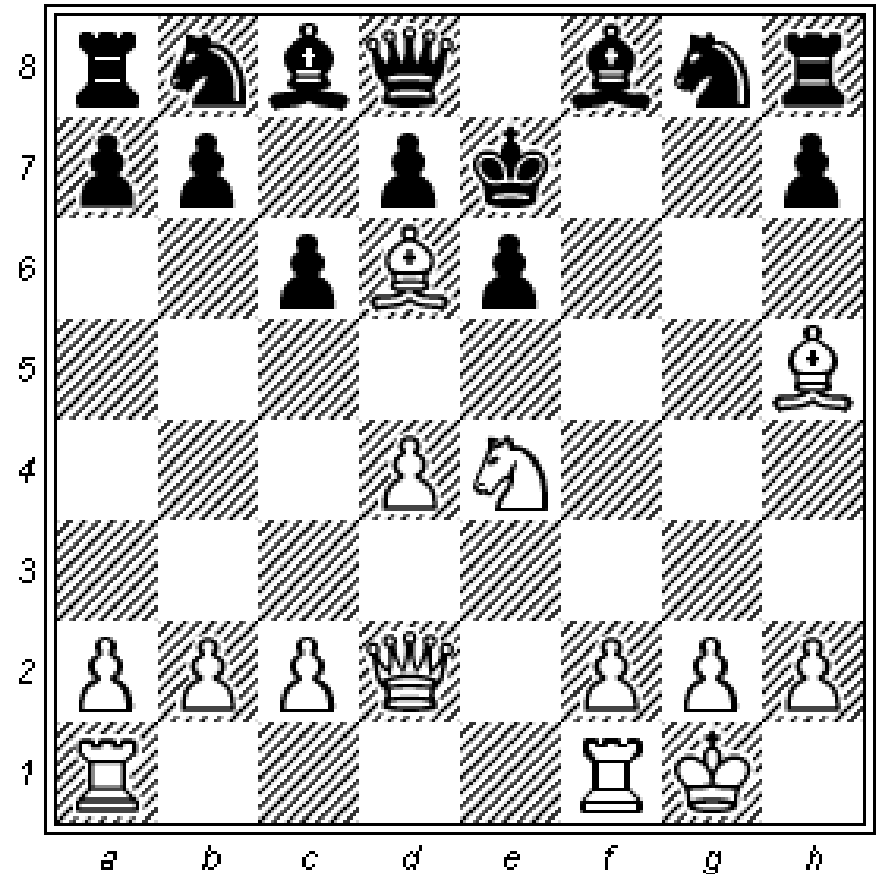
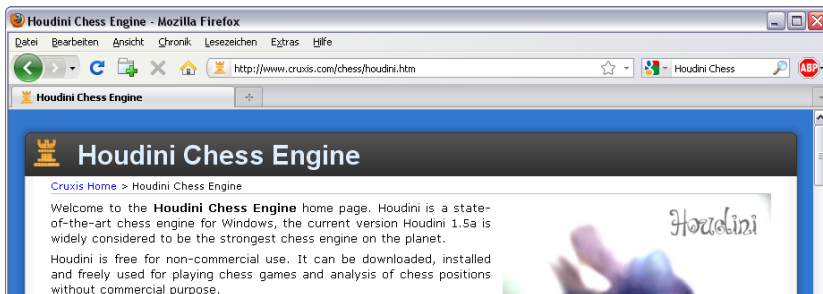


# DokChess gegen Houdini

Houdini 1.5a - DokChess Iter 3  
Buchholz 2011



<http://www.cruxis.com/chess/houdini.htm>



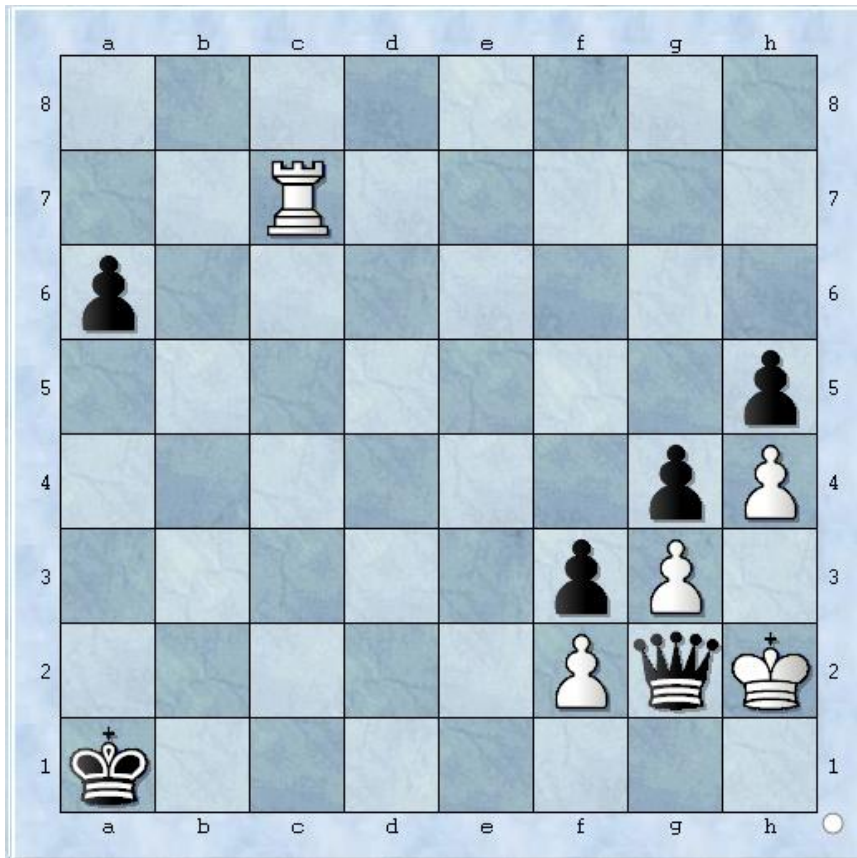
Weiß setzt nach 18 Zügen matt.

# Agenda

## 4

- 0 Die Aufgabe
- 1 Iteration 1: „Der Durchstich“
- 2 Iteration 2: „Das Bauerndiplom“
- 3 Iteration 3: „Der Taktikfuchs“
- 4 Ausblick und Weitere Informationen

# Limitation durch Tiefe (1)



Wei am Zug kann nur den Turm bewegen!

Bei Suchtiefe 2 (2 Halbzge) wrde Wei

1. T c8 – c1 +

nicht spielen, denn durch

1. ... D f1 x c1

verliert wei 5 Materialpunkte.

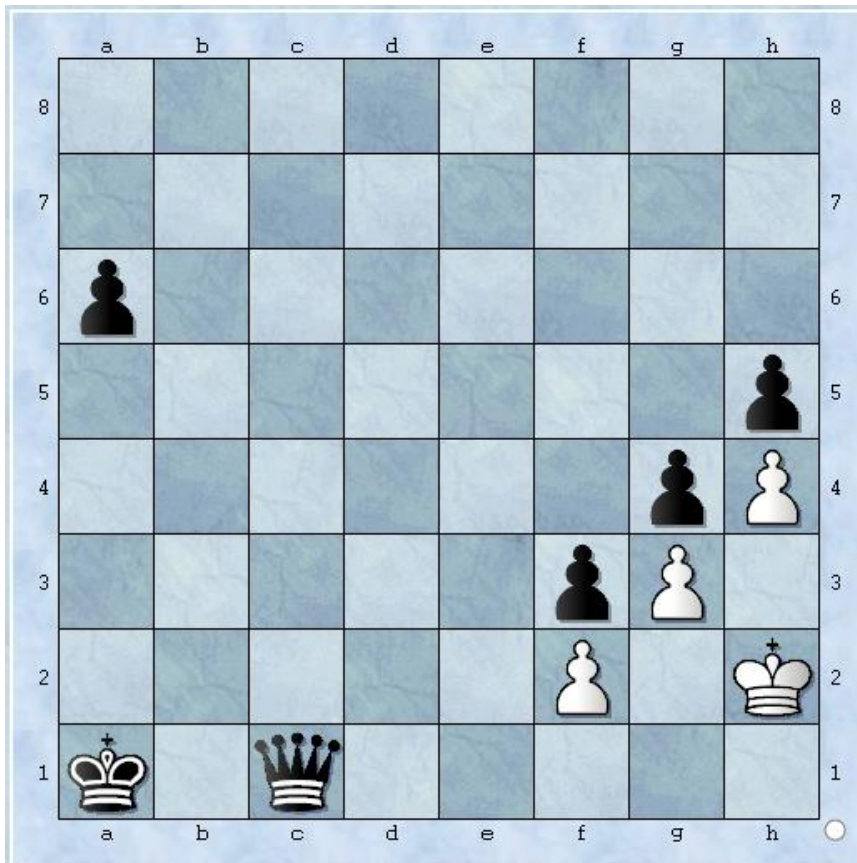
Wegen

1. T c8 – c7

1. ... D f1 – g2#

ist das Spiel dann fr wei zu Ende (Schach matt)

## Limitation durch Tiefe (2)



Aber ...

Tatsächlich ist

1. T c8 – c1 +  
aber die einzige Rettung.

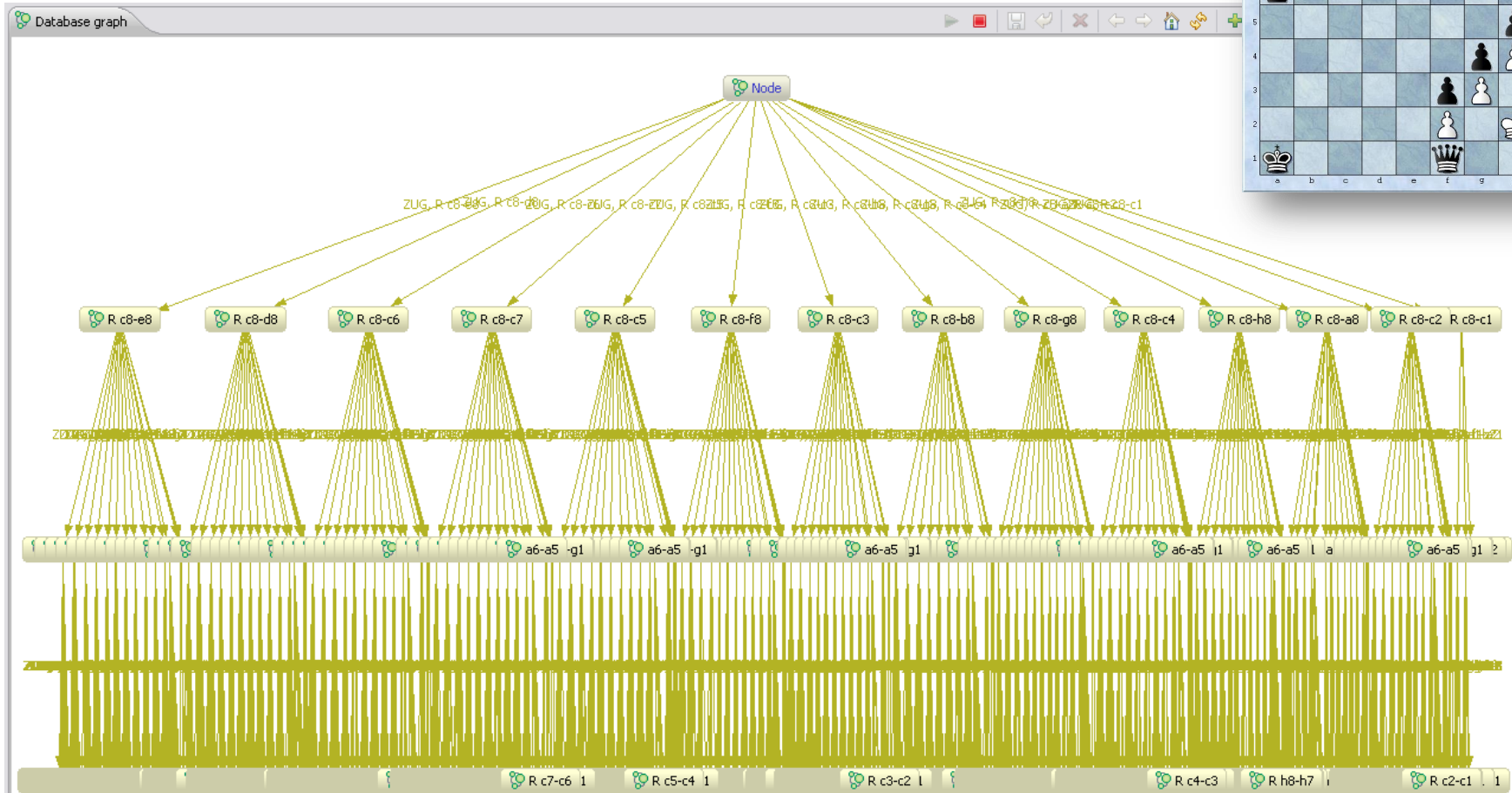
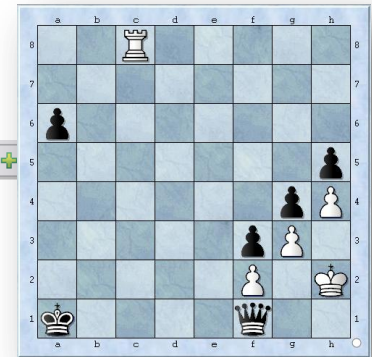
1. ... D f1 x c1

ist erzwungen: das Schach muss erwidert werden, und sonst ist die Dame futsch.

Weiß kann nicht mehr ziehen, steht aber nicht im Schach. Also patt (unentschieden).

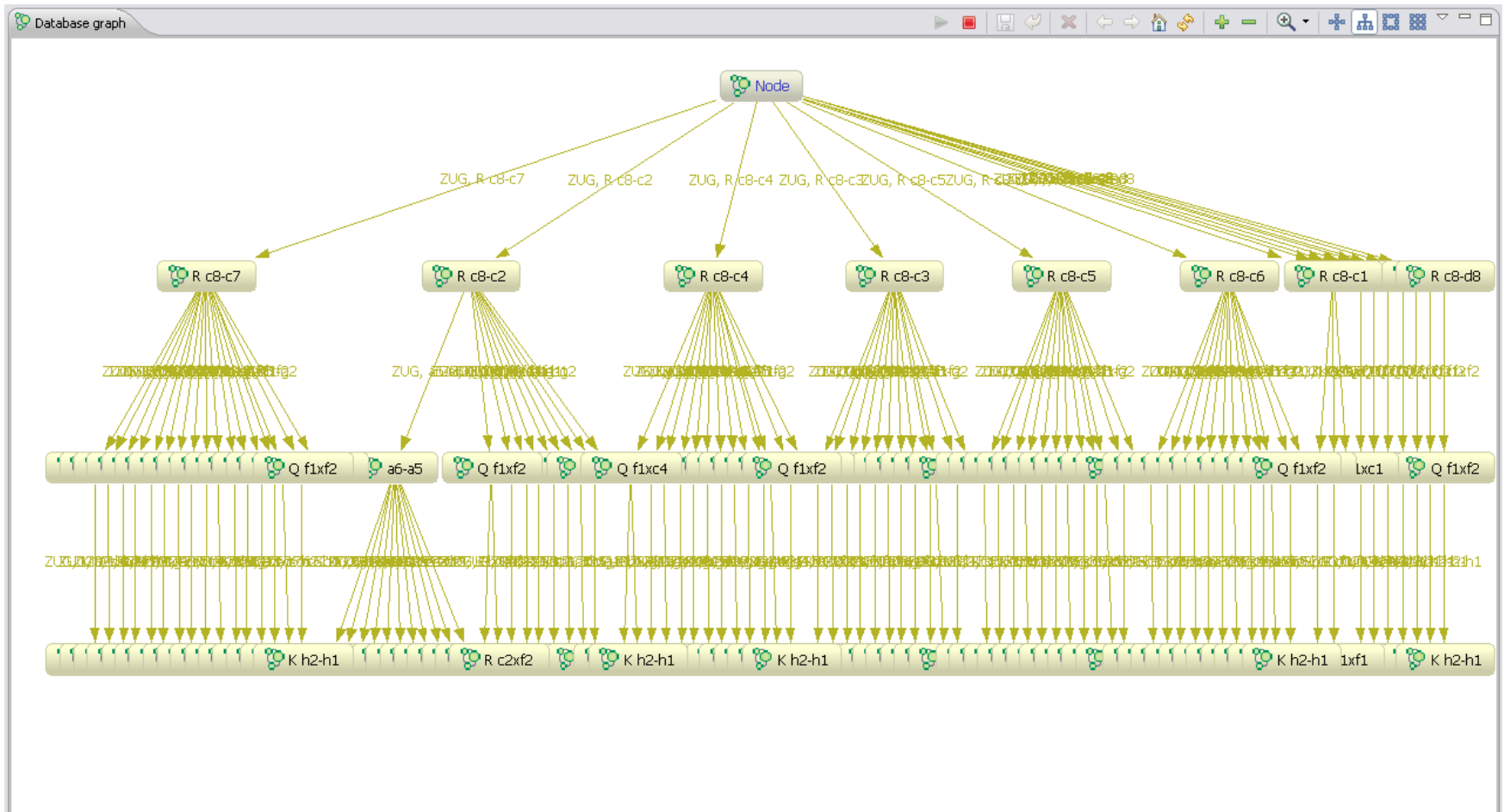
Das findet der Minimax bereits bei Suchtiefe 3!

# Beispiel: Suchbaum Minimax-Algorithmus



Tiefe: 3 Halbzüge, Bewertete Positionen: 2021

## Suchbaum Alpha-Beta-Algorithmus (gleiches Ergebnis)



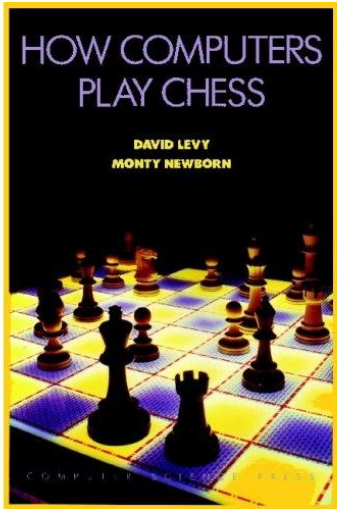
Tiefe: 3 Halbzüge. Bewertete Positionen: 91 (4,5 % von Minimax)

# Nächste Schritte





## (Weitere) Buchempfehlungen



### How Computers Play Chess

von David Levy, Monty Newborn

Taschenbuch: 256 Seiten

Verlag : Ishi Press (2009, Nachdruck von 1990)

Sprache: English

ISBN-13: 978-4871878012



### Einführung in die Schachtaktik

von John Nunn

Taschenbuch: 160 Seiten

Verlag: Gambit Publications (2004)

Sprache: Deutsch

ISBN-13: 978-1904600114



# Fallbeispiel DokChess im Internet

- Architekturüberblick gegliedert nach arc42
- Quelltexte, Links, etc.



Firefox

DokChess: Softwarearchitektur en passant

www.dokchess.de

Sie sind hier: Willkommen



Willkommen DokChess Das Buch Informationen Kontakt Impressum

## Softwarearchitektur en passant

Ein Jahrhunderttraum wie das Fliegen: Eine Maschine, die Menschen im Schach bezwingt. Auch heute noch für viele Entwickler eine faszinierende Aufgabe!

Wie wird das Problem geschickt? Welche wichtigen Entscheidungen sind bei der Umsetzung zu treffen? Mit DokChess lernen Sie das Nötigste, um selbst ein Schachprogramm zu bauen. Und Sie erfahren auf vergnügliche Weise, wie Sie ganz allgemein eine nachvollziehbare, angemessene Softwarearchitektur entwerfen, bewerten und festhalten können.

Demnächst ...

 Vortrag rheinjug 2012 24.05.12

**Objek** Vortrag ObjektForum 2012 11.06.12

Zuletzt ...

 Vortrag JAX

<http://www.dokchess.de/>

# Veranstungshinweis

Stefan Zörner: Wiki ausgedruckt?

10 praxistaugliche Tipps für eine Architekturdokumentation

Herbstcampus | 06.09.2012 | 15:40 - 16:50 Uhr

„Füllen Sie diese tolle Vorlage aus, dann vergessen Sie garantiert nichts.  
Verwenden Sie UML! Ist standardisiert, versteht jeder ...“

Mit solch eines Beraters würdigen Hinweisen wird man in diesem Vortrag verschont. Stattdessen gibt es echtes Erfahrungswissen zum Festhalten einer Software-Architektur. Welche Ziele können damit verfolgt werden? Was gehört unbedingt rein? Welche Werkzeuge und Notationen haben sich bei Erstellung und Pflege bewährt? Und welche bei der Kommunikation im Team und an Dritte? Und vielleicht soll ja wirklich etwas ausgedruckt werden ...



3.– 6. September 2012  
in Nürnberg



# Herbstcampus

Wissenstransfer  
par excellence

**Vielen Dank!**  
**Ich freue mich auf Eure Fragen!**

**Stefan Zörner**

oose Innovative Informatik GmbH :: [sz@oose.de](mailto:sz@oose.de) :: [@StefanZoerner](https://www.instagram.com/StefanZoerner)

**oose.**  
Innovative Informatik