Leistungskurs
Informatik

Schuljahr 12-13
Jahrgang 2011/2012

Projektarbeit

Philipp Schweig

Thema: Schachspiel

Dokumentation

Zuletzt aktualisiert am 2. April 2012 um 18:39

Inhaltsverzeichnis

Anforderungskatalog

Ziel des Projekts

Das Ziel des Projekts war es ein Schachspiel in der Programmiersprache Java für 2 Spieler an einem Rechner zu entwickeln.

Beschreibung der Aufgabenstellung

Zu Beginn soll der Spieler ein Programm vor sich haben, dass ihm ein Schachbrett anzeigt, mit dem er dann mit einem zweiten Spieler eine Schachpartie spielen kann. Dabei soll abwechselnd jeder Spieler einen Zug vollziehen können, in dem er auf eine Figur klickt, die er laut den Schachregeln bewegen könnte und diese dann an eine andere Stelle platziert.

Datenmodell

Variablen, ihre Datentypen und Verwendung – Eine Auswahl

Klasse: Figur

Schachfeld[][] alleFelder:

„alleFelder" referenziert auf die Felder des Schachbrettes

Schachfeld feld:

„feld" referenziert auf das Feld, wo die Figur steht

ImageIcon icon:

„icon" speichert das Bild der Figur

String name:

„name" speichert den Namen der Figur

Position pos:

„pos" speichert die Position der Figur auf dem Schachfeld (den Buchstaben und die Zahl)

int farbe:

„farbe" speichert die ID der Farbe, der die Figur zugehört

boolean besiegt:

„besiegt" speichert ob die Figur besiegt ist oder nicht

Klasse: Position

int buchstabe:

„buchstabe" speichert den Wert des Buchstaben, sie kann die Werte 1-8 annehmen, wobei 1 für A steht und 8 für H

int zahl:

„zahl" speichert den Wert der Zahl, sie kann die Werte 1-8 annehmen

boolean angriff:

„angriff" speichert, ob die Position eine Angriffsposition ist

boolean spielzug:

„spielzug" speichert, ob die Position ein Spielzug ist

Schnittstellenbeschreibung

Bedienung des Programms

Das Programm ist über zwei Ansichten bedienbar. Einmal die Startansicht (siehe Anhang, Abbildung 1), bei der man die Namen der Spieler eingeben kann und einmal die Spielansicht. In dieser gibt es dann ein Schachbrettbereich, wo der Spieler, der gerade am Zug ist seine Schachfiguren (nur die, die sich laut den Regeln und der aktuellen Spielsituation auch bewegen können) auswählen und anschließend versetzen kann. Der Spieler klickt dazu auf einen Button der aktiviert ist (grün gekennzeichnet, siehe Anhang, Abbildung 2), worauf dann die möglichen Züge angezeigt werden. Der blau gekennzeichnete Button wird wieder deaktiviert. Die möglichen Züge und die daraus entstehenden Angriffszüge werden auch durch Buttons markiert (blau bzw. rot markiert, siehe Anhang, Abbildung 3). Der Spieler soll aber auch die Möglichkeit haben, den Zug abzubrechen, in dem er einfach in ein nicht aktiviertes Feld klickt. Wenn der Spieler den Zug dann durch Klicken auf einen Button der möglichen Züge oder Angriffszüge vollzogen hat, ist der andere Spieler an der Reihe und das Ganze geht wieder von vorne los.

Funktionsweise der Code-Logik

Das Programm besteht aus einem Frame, welches in ein Startpanel und ein Spielpanel gegliedert ist. Der Frame besitzt ein CardLayout, welches es ermöglicht, die Panels, die auf diesem sind, gezielt anzeigen zu lassen. Ein CardLayout funktioniert wie übereinandergelegte Karteikarten, sodass immer nur das oberste Panel angezeigt wird. Auf dem Spielpanel sitzt das Schachbrett (erbend von einem Panel).

Das Schachbrett besteht aus mehreren Schachfeldern, die sich wiederum aus einem Button und einem Label zusammensetzen. Wenn ein Schachfeld aktiviert ist bedeutet das, dass der Button sichtbar und das Label versteckt ist, ist das Schachfeld jedoch deaktiviert, dann ist das Label sichtbar und der Button unsichtbar. Auf beiden Elementen wird das Bild angezeigt. Außerdem enthält das Schachfeld auch das aktuelle Figur-Objekt. Versetzt der Spieler die Figur, meldet diese sich von dem Schachfeld ab und bei dem neuen an. Somit ist die Figur das Händlerobjekt, von der alles ausgeht.

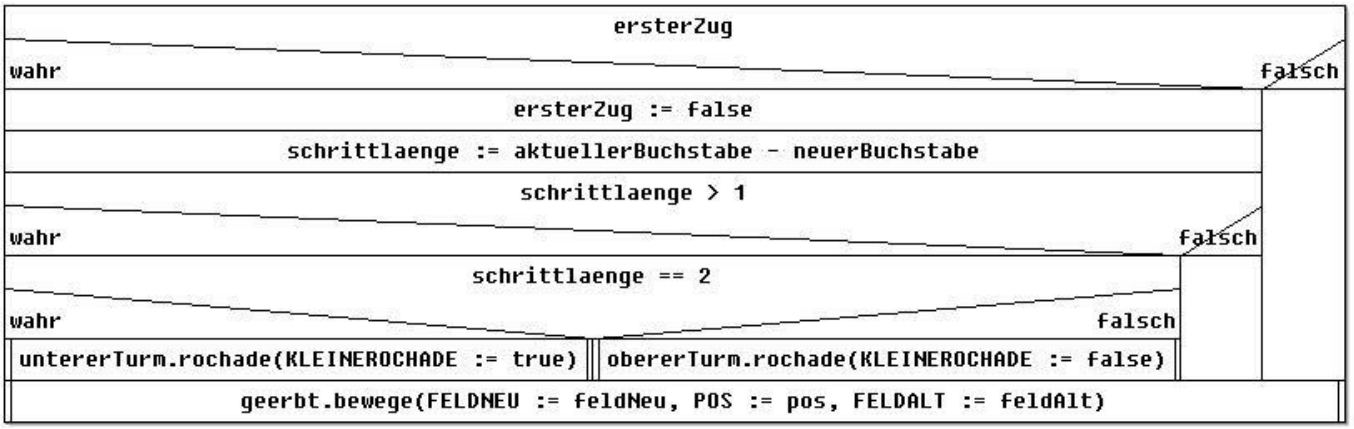
Beschreibung der Module

Struktogramme

Beispiel: F_Koenig : bewege

Rückgabetyp: Keiner Lokale Variablen: Keine

Parameter: Konstanten: Keine
pos: Position



Beispiel: F_Turm : rochade

Rückgabetyp: Keiner Lokale Variablen:
pos: Zeichenkette

Parameter:
kleineRochade: Wahrheitswert Konstanten: Keine

UML-Diagramm

siehe Anhang

Verlaufsbeschreibung

07. Januar 2012

- Klassendateien angelegt

  Position, Schachfigur, SF_Bauer, SF_Dame, SF_Koenig, SF_Laeufer, SF_Springer, SF_Turm, Spiel, Schachbrett

13. Januar 2012

- alle Klassen die von Figur erben, nach F_#name# umbenannt

20. Januar 2012

- Schachbrett Panel in GridBagLayout geändert
- Syntaxfehler behoben => weißes Panel als Resultat -.-

23. Januar 2012

- Umdenken: Nutzen von Swing anstatt AWT
- Daten von 20.01.12 über Bord geworfen und auf die Daten vom 13.01.12 gesetzt
- wiederholt Anordnung des Schachbrettes durch ein GridBagLayout
- Farben der Schachfelder gestaltet
- Bauern werden aufs Schachfeld geladen

02. Februar 2012

- Figur-Klasse als abstrakt deklariert

  alle Klassen die von Figur erben, müssen alle abstrakten Methoden definieren
- Damen werden aufs Schachfeld geladen
- Spiel wird nun mit Globaler Instanz instanziiert
- Schachfigur Klasse gelöscht, da man F_#name# auf Figur setzen kann (dank objektorientierte Vererbung)
- Dreh- und Angelpunkt ist jetzt nicht mehr das Schachfeld, sondern die Figur

03. Februar 2012

- Ordner „Images" angelegt
- erstes Figurenbild hinzugefügt

04. – 05. Februar 2012

- kleine Änderungen

- Regeln den einzelnen Figuren hinzugefügt
  Rochade angelegt

06. Februar 2012

- Klassen GUIObjektDaten und Koordinaten angelegt
- Figuren können sich nun bewegen

08. Februar 2012

- Rochaderegel falsch => muss noch behoben werden

10. Februar 2012

- Spiel ist erst einmal spielbar, es fehlen aber noch ein paar Regeln

24. Februar 2012

- Tools-Klasse angelegt
- Rochade fertiggestellt
- IBauer-Interface angelegt
- Bauern können als Dame ausgetauscht werden, wenn sie am Ende angekommen sind
  Auswahl für alle Figuren folgt
- nach Spielende Möglichkeit zum zurückgehen hinzugefügt
- bei Figurenauswahl nur noch die anklickbar, die auch versetzt werden können

25. Februar 2012

- Figurenauswahl, wenn Bauer am Ende angekommen ist, hinzugefügt

27. Februar 2012

- „NullPointerExeption" behoben, wenn Bauer ausgetauscht und die ausgetauschte Figur gleich im Anschluss geschlagen wurde

Beschreibung von Schwierigkeiten und Fehlern

Schwierigkeiten während meiner Entwicklung des Programmes, waren zum einen das Layouten des Schachbretts mit den Schachfeldern, das ich durch das GridBayLayout gelöst habe und der Zugriff von einer Figur auf das Spielobjekt, das ich dadurch gelöst habe, in dem ich das Spielobjekt global gemacht habe, bzw. als Singleton-Objekt. Wikipedia definiert Singleton zum Zeitpunkt der Erstellung dieser Dokumentation wie folgt:

> *Das Singleton (auch Einzelstück genannt) ist ein in der Softwareentwicklung eingesetztes Entwurfsmuster und gehört zur Kategorie der Erzeugungsmuster (engl. Creational Patterns). Es verhindert, dass von einer Klasse mehr als ein Objekt erzeugt werden kann. Dieses Einzelstück ist darüber hinaus üblicherweise global verfügbar.*

Quelle: http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster)

Beurteilung des Ergebnisses

Vergleich mit dem geplanten Projekt

Das entstandene Projekt ist im Großen und Ganzen genau meiner Vorstellung treu geworden. Es sind nur Kleinigkeiten, wie die Eingabe der Namen zu Beginn oder die Anzeige dieser auf der Seite des Schachbrettes dazugekommen.

Fehleranalyse

In Anbetracht meiner Vorstellungen am Anfang und dem Resultat, gibt es nur ein Problem, dass der König nach aktuellem Stand in Schach ziehen kann, was laut den Schachregeln aber nicht geht. So wie ich es versucht habe zu lösen, kam es zu einer Endlosschleife, da die Funktion der Klasse F_Koenig „zuege", wo ich prüfe, auf welche Positionen die Figur sich setzen kann und die Funktion der Klasse Spieler „alleMoeglichenZuege" sich rekursiv immer wieder aufrufen würden.

Anhang

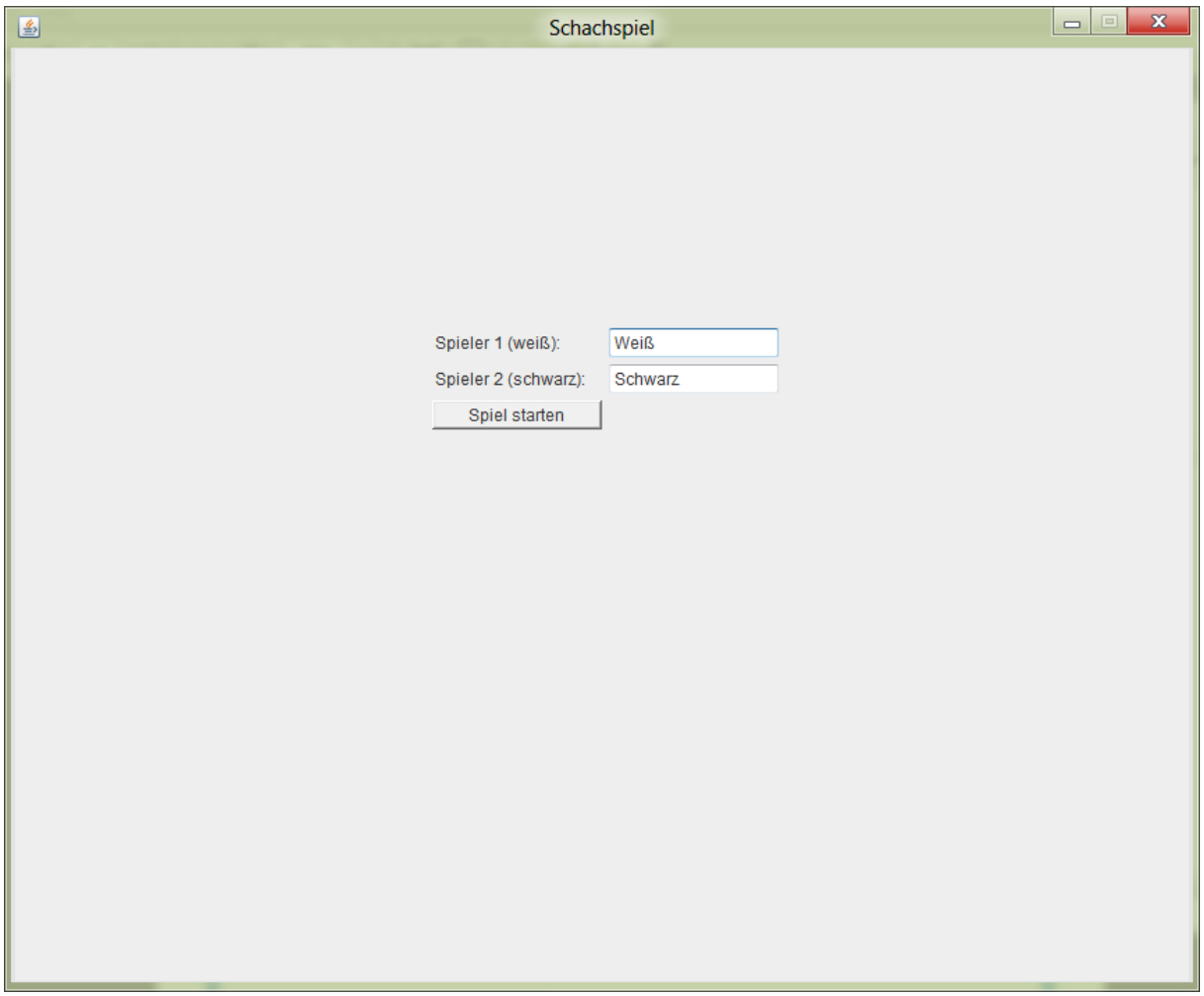

Schachspiel
Spieler 1 (weiß):
Weiß
Spieler 2 (schwarz):
Schwarz
Spiel starten

Abbildung 1

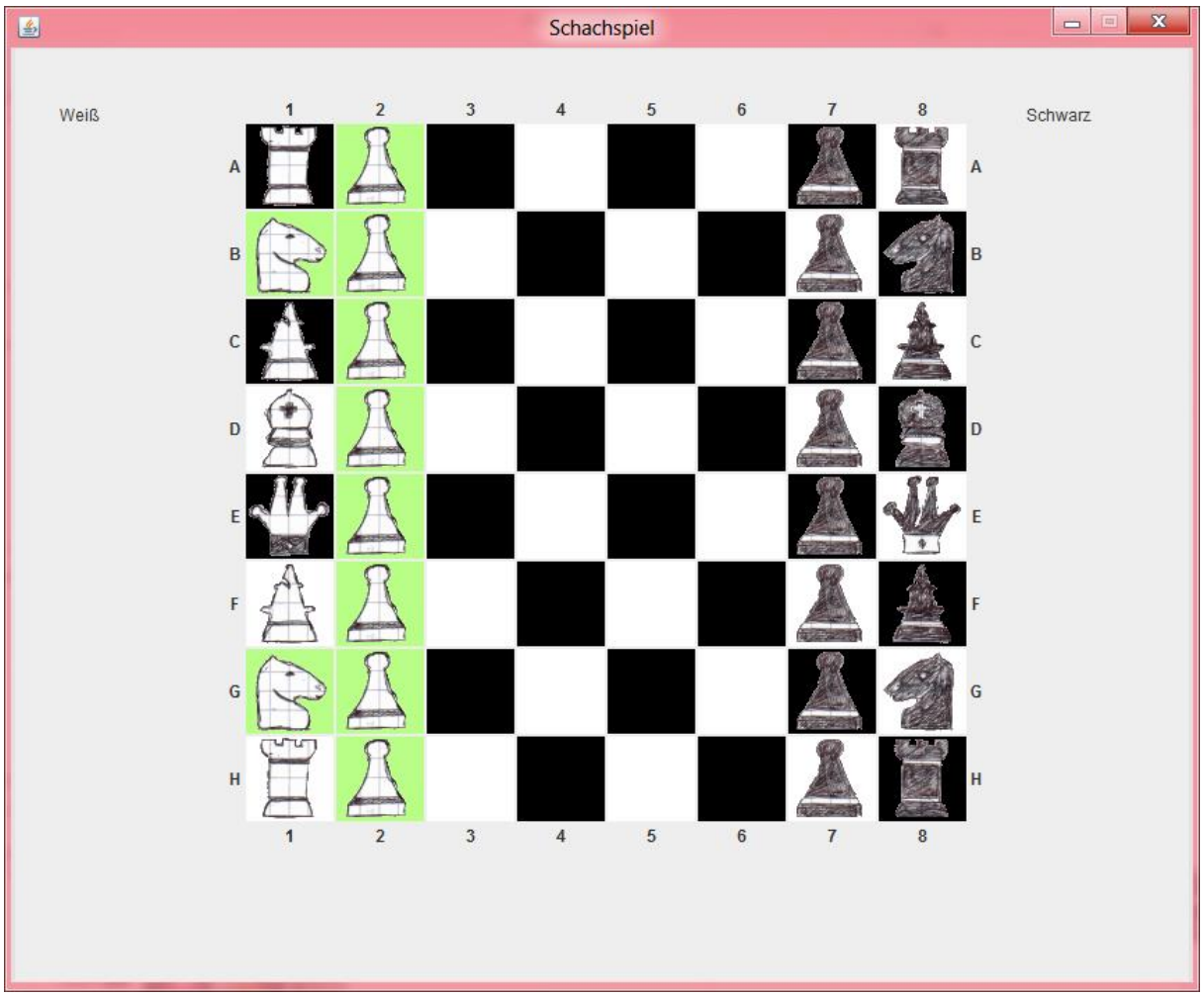


Abbildung 2

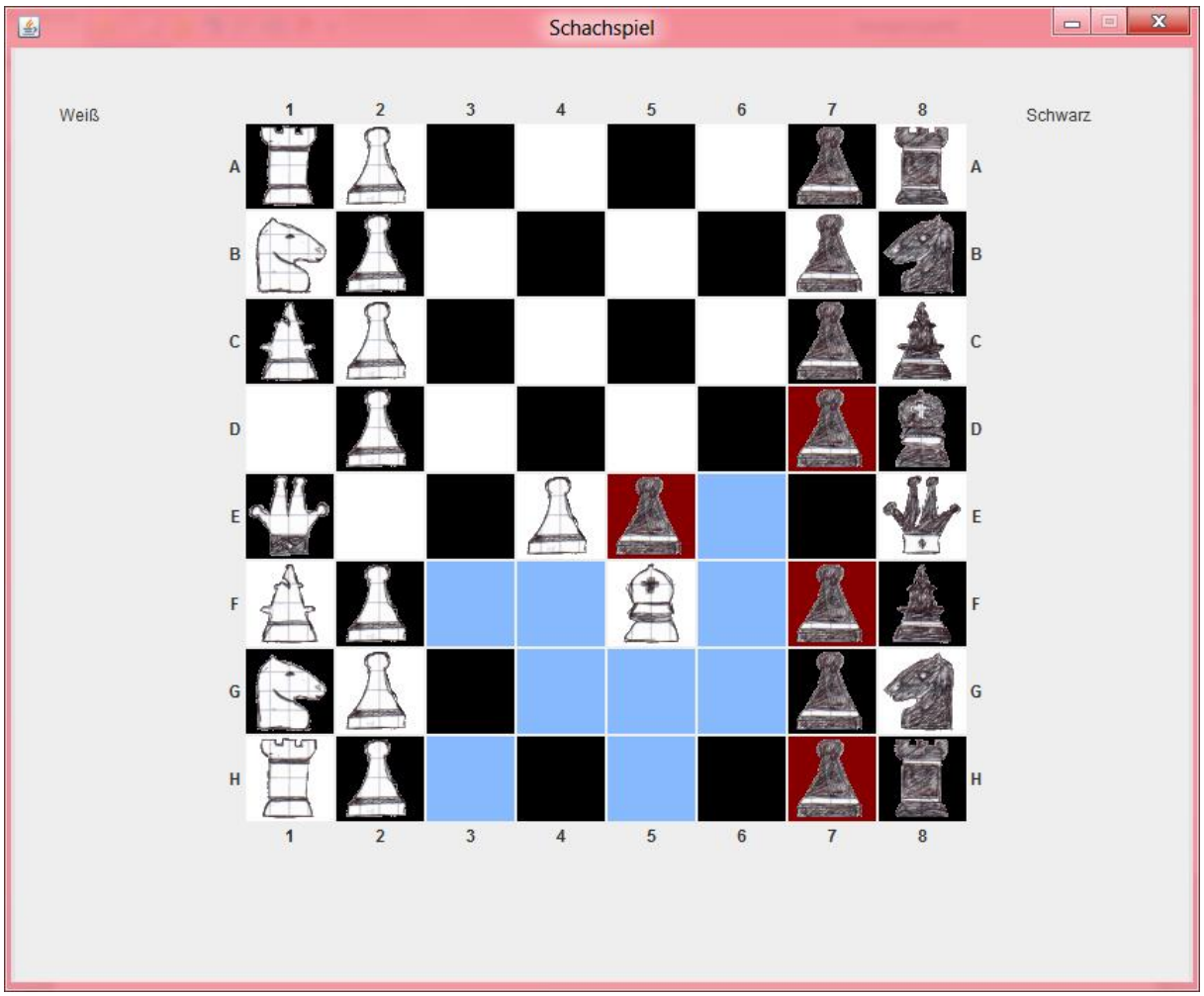
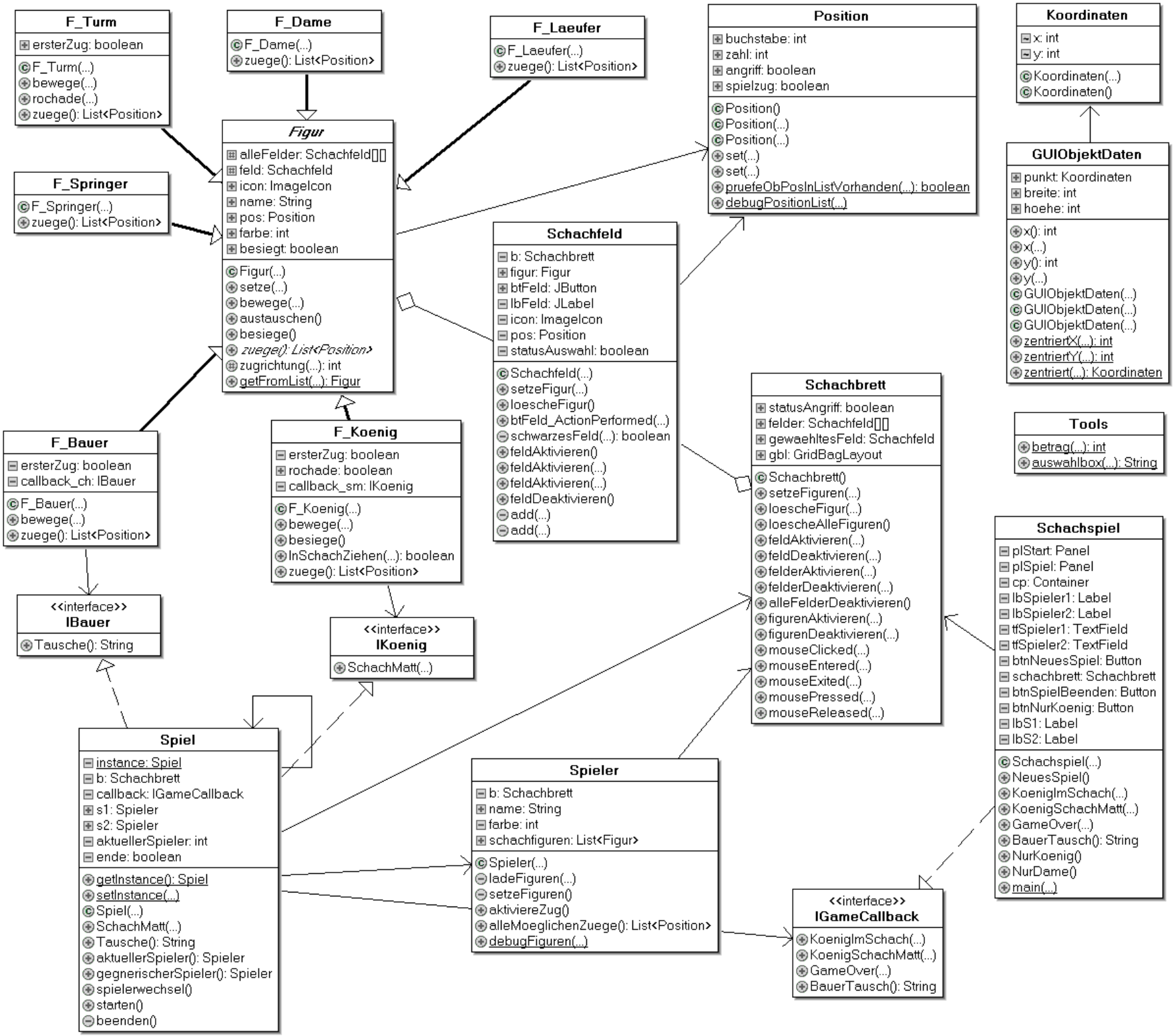


Abbildung 3


F_Turm
ersterZug: boolean
F_Turm(...)
bewege(...)
rochade(...)
zuege(): List<Position>
F_Dame
F_Dame(...)
zuege(): List<Position>
F_Laeufer
F_Laeufer(...)
zuege(): List<Position>
Position
buchstabe: int
zahl: int
angriff: boolean
spielzug: boolean
Position()
Position(...)
Position(...)
set(...)
set(...)
pruefeObPosInListVorhanden(...): boolean
debugPositionList(...)
Koordinaten
x: int
y: int
Koordinaten(...)
Koordinaten()
GUIObjektDaten
punkt: Koordinaten
breite: int
hoehe: int
x(): int
x(...)
y(): int
y(...)
GUIObjektDaten(...)
GUIObjektDaten(...)
GUIObjektDaten(...)
zentriertX(...): int
zentriertY(...): int
zentriert(...): Koordinaten
F_Springer
F_Springer(...)
zuege(): List<Position>
Figur
alleFelder: Schachfeld[][]
feld: Schachfeld
icon: ImageIcon
name: String
pos: Position
farbe: int
besiegt: boolean
Figur(...)
setze(...)
bewege(...)
austauschen()
besiege()
zuege(): List<Position>
zugrichtung(...): int
getFromList(...): Figur
Schachfeld
b: Schachbrett
figur: Figur
btFeld: JButton
lbFeld: JLabel
icon: ImageIcon
pos: Position
statusAuswahl: boolean
Schachfeld(...)
setzeFigur(...)
loescheFigur()
btFeld_ActionPerformed(...)
schwarzesFeld(...): boolean
feldAktivieren()
feldAktivieren(...)
feldAktivieren(...)
feldDeaktivieren()
add(...)
add(...)
Schachbrett
statusAngriff: boolean
felder: Schachfeld[][]
gewaehltesFeld: Schachfeld
gbl: GridBagLayout
Schachbrett()
setzeFiguren(...)
loescheFigur(...)
loescheAlleFiguren()
feldAktivieren(...)
feldDeaktivieren(...)
felderAktivieren(...)
felderDeaktivieren(...)
alleFelderDeaktivieren()
figurenAktivieren(...)
figurenDeaktivieren(...)
mouseClicked(...)
mouseEntered(...)
mouseExited(...)
mousePressed(...)
mouseReleased(...)
Tools
betrag(...): int
auswahlbox(...): String
F_Bauer
ersterZug: boolean
callback_ch: IBauer
F_Bauer(...)
bewege(...)
zuege(): List<Position>
F_Koenig
ersterZug: boolean
rochade: boolean
callback_sm: IKoenig
F_Koenig(...)
bewege(...)
besiege()
InSchachZiehen(...): boolean
zuege(): List<Position>
Schachspiel
plStart: Panel
plSpiel: Panel
cp: Container
lbSpieler1: Label
lbSpieler2: Label
tfSpieler1: TextField
tfSpieler2: TextField
btnNeuesSpiel: Button
schachbrett: Schachbrett
btnSpielBeenden: Button
btnNurKoenig: Button
lbS1: Label
lbS2: Label
Schachspiel(...)
NeuesSpiel()
KoenigImSchach(...)
KoenigSchachMatt(...)
GameOver(...)
BauerTausch(): String
NurKoenig()
NurDame()
main(...)
<<interface>>
IBauer
Tausche(): String
<<interface>>
IKoenig
SchachMatt(...)
Spiel
instance: Spiel
b: Schachbrett
callback: IGameCallback
s1: Spieler
s2: Spieler
aktuellerSpieler: int
ende: boolean
getInstance(): Spiel
setInstance(...)
Spiel(...)
SchachMatt(...)
Tausche(): String
aktuellerSpieler(): Spieler
gegnerischerSpieler(): Spieler
spielerwechsel()
starten()
beenden()
Spieler
b: Schachbrett
name: String
farbe: int
schachfiguren: List<Figur>
Spieler(...)
ladeFiguren(...)
setzeFiguren()
aktiviereZug()
alleMoeglichenZuege(): List<Position>
debugFiguren(...)
<<interface>>
IGameCallback
KoenigImSchach(...)
KoenigSchachMatt(...)
GameOver(...)
BauerTausch(): String