Produktdokumentation Vier Gewinnt - untitled0815 – 02.11.2012

Alexander Busch Nora Herentrey Björn List Johannes Riedel Henny Selig Sascha Ulbrich

Produktdokumentation

Vier Gewinnt

- untitled0815 -

Inhaltsverzeichnis

[A. Benutzerdokumentation 2](#_Toc339565908)

[A.1. Grundlagen 2](#_Toc339565909)

[A.2. Erste Schritte 2](#_Toc339565910)

[A.3. Konfiguration 3](#_Toc339565911)

[[A] KI gegen Server 3](#_Toc339565912)

[[B] Spieler gegen KI 3](#_Toc339565913)

[[C] Spielstand laden 4](#_Toc339565914)

[A.4. Das Spiel 4](#_Toc339565915)

[[A] KI gegen Server 4](#_Toc339565916)

[[B] Spieler gegen KI 4](#_Toc339565917)

[[C] Spielstand laden 4](#_Toc339565918)

[A.5. Funktionsmenü 4](#_Toc339565919)

[Spielsteuerung 4](#_Toc339565920)

[Hilfe 4](#_Toc339565921)

[B. Entwicklerdokumentation 5](#_Toc339565922)

[B.1. Ziele und äußerer Rahmen 5](#_Toc339565923)

[B.2. Zielarchitektur 6](#_Toc339565924)

[B.2.1. Model-View-Viewmodel als grundsätzliches Architekturkonzept unseres Programms 6](#_Toc339565925)

[B.2.2. Aufgaben der einzelnen Komponenten 7](#_Toc339565926)

[B.2.2.5. Tests 8](#_Toc339565927)

[B.3. Programmablauf 8](#_Toc339565928)

[B.4. Anhang 8](#_Toc339565929)

[B.4.1. Klassendiagramm 8](#_Toc339565930)

[B.4.2. Entity-Relationship Diagramm 8](#_Toc339565931)

[B.4.3. Javadoc 9](#_Toc339565932)

[B.4.4 Sequenzdiagramme 9](#_Toc339565933)

[C. Projektdurchführung 10](#_Toc339565934)

[C.1. Entwicklungsmodell und Testkonzept 10](#_Toc339565935)

[C.1.1. Das Entwicklungsmodell 10](#_Toc339565936)

[C.2. Javadoc 12](#_Toc339565937)

[3.4.1. Javadoc im Projekt 12](#_Toc339565938)

[3.4.2. Javadoc Anleitung 12](#_Toc339565939)

[C.3. Git 15](#_Toc339565940)

[C.4. Databinding 16](#_Toc339565941)

# Benutzerdokumentation

## Grundlagen

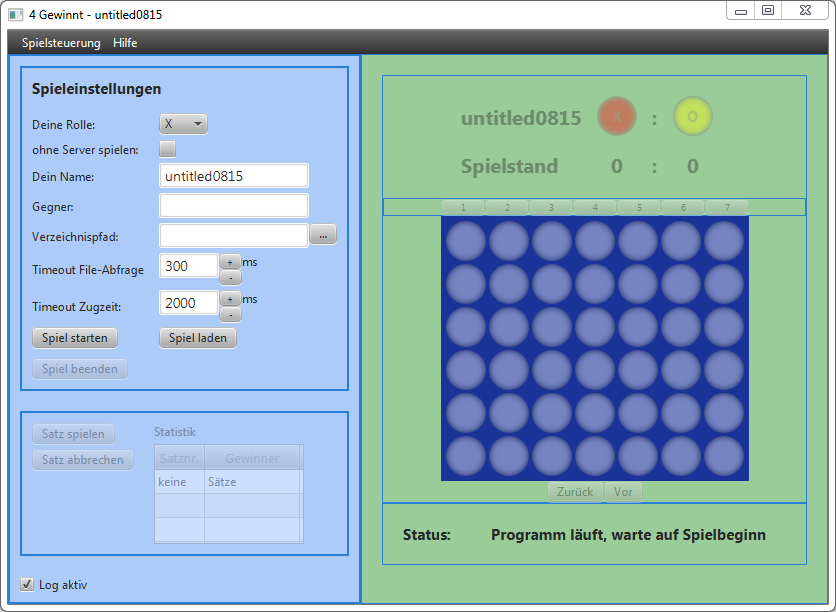
Bei dem vorliegenden Javaprojekt der Gruppe untitled0815 bestehend aus Alexander Busch, Nora Herentrey, Björn List, Johannes Riedel, Henny Selig, Sascha Ulbrich handelt es sich um eine Anwendung, welche es ermöglicht, das bekannte Gesellschaftsspiel Vier Gewinnt in verschiedenen Modi zu spielen.

Dem Nutzer wird ermöglicht entweder gegen den Computer zu spielen, einen vergangenen Spielstand zu laden oder den Computer in Kombination mit einem Server gegen einen anderen Spieler oder einen weiteren Computer spielen zu lassen.

## Erste Schritte

Der Software-Agent steht als ausführbarer Java-Export (\*.jar) zur Verfügung. Um die Anwendung zu starten, öffnet man diese mit einem Doppelklick auf XXXXXXXXX.jar. Hierbei ist darauf zu achten, dass auf dem PC, welcher die Anwendung ausführt eine Java Runtime Environment der Version 7 oder höher installiert ist.[[1]](#footnote-1)

Nach dem Start sollte der Benutzer die Spieloberfläche, wie nachfolgend dargestellt, sehen.



## Konfiguration

Bevor das Spiel beginnt muss der Benutzer die Wahl treffen, ob die KI gegen den Server spielt [A], der Spieler gegen die KI spielt [B] oder ein Spielstand geladen werden soll [C].

### [A] KI gegen Server

1. Deine Rolle: Wählen Sie Ihre Rolle aus (X – Rot oder O – Gelb).
2. Ohne Server spielen: Lassen Sie das Feld „ohne Server spielen“ deaktiviert.
3. Dein Name: Tragen Sie Ihren Namen ein.
4. Gegner: Tragen Sie den gegnerischen Namen ein.
5. Verzeichnispfad: Tragen Sie den Verzeichnispfad in welchem sich die Serverdatei befindet ein oder wählen Sie es über einen Klick auf „…“ aus.
6. Timeout File-Abfrage: Tragen Sie den Wert ein, in welchen Abständen nach neuen Serverdaten abgefragt werden darf (oder erhöhen/mindern sie mit + oder -).
7. Timeout Zugzeit: Tragen Sie den Wert ein, wie lange der Agent Zeit hat, um seinen Zug durchzuführen (oder erhöhen/mindern sie mit + oder -).
8. Log aktiv: Wählen Sie aus, ob die Spielaktionen dokumentiert werden sollen.
9. Spiel starten: Drücken Sie nun „Spiel starten“ um mit dem Spiel zu beginnen. Die Einstellungen werden übernommen und fixiert.

### [B] Spieler gegen KI

1. Deine Rolle: Wählen Sie ihre Rolle (X – Rot oder O – Gelb).
2. Ohne Server spielen: Wählen Sie das Feld an.
3. Dein Name: Tragen Sie ihren Namen ein.
4. Timeout Zugzeit: Tragen Sie den Wert ein, wie lange der Agent Zeit hat, um seinen Zug durchzuführen (oder erhöhen/mindern sie mit + oder -).
5. Log aktiv: Wählen Sie aus, ob die Spielaktionen dokumentiert werden sollen.
6. Spiel starten: Drücken Sie nun „Spiel starten“ um mit dem Spiel zu beginnen. Die Einstellungen werden übernommen und fixiert.

### [C] Spielstand laden

1. Klicken Sie auf die Schaltfläche „Spiel laden“
2. Wählen Sie ein entsprechendes Spiel aus.
3. Wählen Sie die Option Wiederholung manuell abspielen, wenn Sie das Spiel Schritt für Schritt wiederholen lassen möchten.

## Das Spiel

Je nachdem welcher Spielmodi gewählt wurde unterscheidet sich die Spielsteuerung.

### [A] KI gegen Server

1. Satz spielen: Klicken sie nun auf „neuen Satz spielen“ um den Agenten in Bereitschaft zu versetzen.
2. Warten Sie, bis die Züge getätigt wurden und der Satz beendet wird.
3. Falls der Satz nicht vorzeitig durch „Satz abbrechen“ unterbrochen wurde erscheint nach Spielende der Dialog „Gewinner bestätigen“ in welchem mittels Drop-Down Menü der Gewinner bestätigt oder geändert werden kann.
4. Wählen Sie im Anschluss, ob Sie einen weiteren Satz spielen möchten oder das Spiel beendet werden soll.

### [B] Spieler gegen KI

1. Neuen Satz spielen: Klicken Sie nun auf „neuen Satz spielen“ um den Satz zu beginnen.
2. Um einen Stein zu platzieren, klicken Sie auf die jeweilige Position, Spalte oder auf den Button mit der Spaltennummer.
3. Falls der Satz nicht vorzeitig durch „Satz abbrechen“ unterbrochen wurde erscheint nach Spielende der Dialog „Gewinner bestätigen“ in welchem mittels Drop-Down Menü der Gewinner bestätigt oder geändert werden kann.
4. Wählen Sie im Anschluss, ob Sie einen weiteren Satz spielen möchten oder das Spiel beendet werden soll.

### [C] Spielstand laden

## Funktionsmenü

### Spielsteuerung

Unter dem Menüpunkt „Spielsteuerung“ erreicht man, wie auch auf der sichtbaren Oberfläche die Auswahlpunkte „Spiel starten“, „Spiel laden“, „Spiel beenden“ und „Programm schließen“

### Hilfe

Unter dem Menüpunkt „Hilfe“ hat der Nutzer die Möglichkeit den Log anzeigen zu lassen oder die Spielanleitung zu öffnen.

# Entwicklerdokumentation

## Ziele und äußerer Rahmen

**Ziel** des Projektes ist die Implementierung eines „4-gewinnt“-Agenten.

Die verwendete Programmiersprache ist **Java** (Version JRE 1.7). Entsprechend wird objektorientiert gearbeitet. Der Import der Programmbestandteile in eclipse auf anderen PCs wird zur Verfügung gestellt, ebenso wird das Programm zum Spielen in einer ausführbaren JAR-Datei übergeben.

**Funktionen** des Spiels sind das automatische Spielen gegen ein anderes Programm über den Server. Zusätzlich können vergangene Spiele geladen werden und Schritt für Schritt rekonstruiert werden. Als Zusatzfunktion kann ein Benutzer gegen das Programm spielen.

Eine Benutzeroberfläche (im folgenden **GUI**) für das Spielen wird entsprechend bereit gestellt. Dort kann der Benutzer den Spielmodus wählen. Sämtliche Eingaben und Ausgaben erfolgen über dieses GUI. Dazu zählen die Einstellungen aus dem Projektauftrag wie etwa Zugzeiten, Pfade und Beginn und Abbruch eines neues Spiels oder Satzes. Die Realisierung des GUI erfolgt über Java Fx.

Ebenso besteht eine **Datenbankanbindung**, über die auch vorherige Spielabläufe und –stände, sowie alle weiteren funktionsrelevanten Daten gespeichert und wieder geladen werden können. Als Datenbank wird eine HSQLDB genutzt. Einige Beispieldaten werden zur Verfügung gestellt.

Im Rahmen der Entwicklung entstehen ebenfalls eine Entwicklungsdokumentation und eine Benutzerdokumentation:

* Es wird im Quellcode dokumentiert, Modelle graphisch dargestellt, sowie eine testuelle Dokumentation verfasst
* Jeder ist dafür verantwortlich, den von ihm erstellten Part der Software ausreichend zu dokumentieren. Zusätzlich gibt es eine Kontrolle der Dokumentationstätigkeiten im Rahmen der Projektorganisation.
* Dokumentationssprache ist deutsch.

Das Projekt wird als solches nach dem Spiralmodell durchgeführt. Für die Ablage und Verwaltung des Quellcodes, sowie sämtlicher anderer Dokumente zum Projekt wird GIT genutzt. Dazu wird ein Projekt unter Github.com angelegt.

Im Rahmen des Projektes werden weitere Vertiefungsthemen bearbeitet, dazu zählen:

* Einarbeitung in JavaFX
* Einarbeitung und Nutzung des MVVM-Konzepts
* Überblick gelangen über die Nutzung von GIT
* Das Spiralmodell als Entwicklungsmodell
* Nutzung der Dokumentationsmöglichkeiten von Java

## Zielarchitektur

### Model-View-Viewmodel als grundsätzliches Architekturkonzept unseres Programms

#### Model-View-Controller als Ausgangspunkt

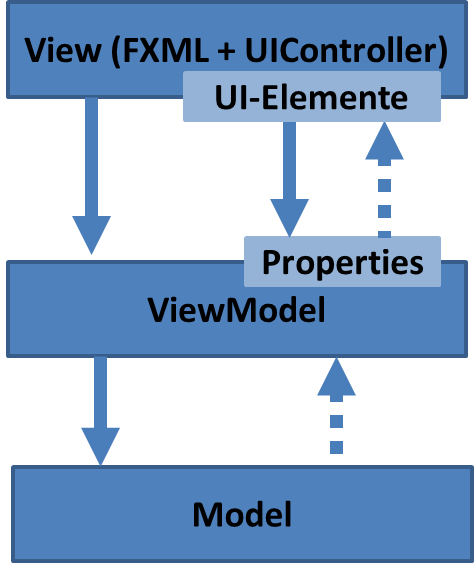
Als Entwurfsmuster für Benutzeroberflächen ist das Model-View-Controller (MVC)-Entwurfsmuster sehr verbreitet. Es basiert auf drei Komponenten, welche die Präsentation von den Daten und der Interaktion trennen.

Die Komponente Model beinhaltet das Datenmodell und informiert die View, falls es Änderungen gibt. Die View reagiert auf Änderungen im Model und stellt die Informationen in der Benutzeroberfläche dar. Aktionen, die von der Benutzeroberfläche kommen nimmt der Controller entgegen und verändert beispielsweise das Model.

MVC ist allerdings nicht optimal für das zu erstellende Programm geeignet, da keine Schicht zwischen View und Model existiert. Somit ist eine Umsetzung nur nach diesem Entwurfsmuster schwierig, da die Benutzeroberfläche nicht direkt mit dem Datenmodell arbeiten, sondern ein extra UI-Datenmodell erstellt werden soll um das reine Datenmodell von der Aufbereitung für das UI, der Bereitstellung von Properties, zu trennen. Deshalb haben wir nach einer ersten Umsetzung nach MVC unsere Architektur auf das im Folgenden beschriebene MVVM-Modell umgestellt.

#### Model-View-Viewmodel als genutzte Weiterentwicklung

Als passende Alternative hat sich das Model-View-ViewModel (MVVM)-Entwurfsmuster angeboten, das eine bessere Trennung von UI-Design und UI-Logik ermöglicht.

Das Model und die View sind dabei mit denen des MVC-Entwurfsmusters vergleichbar. Die View kennt dabei nur das ViewModel und nicht das Model. Außerdem befindet sich im View keine Ablauflogik, diese wird vom ViewModel übernommen, das Properties bereitstellt, die von der View mittels Data Binding genutzt werden. Somit besitzt das ViewModel keine Kenntnis darüber, von welchen Views es referenziert wird. Es greift lediglich auf das Model zu und bereitet dessen Daten auf und reagiert auf Aktionen im UI. Damit erweitert es die Funktionalität des Controllers vom MVC-Entwurfsmuster um die Logik zum Aufbereiten der Daten für die View. Kritik wird am MVVM-Entwurfsmuster vor allem geübt, weil das ViewModel zu viele Aufgaben zu erledigen hat, da es einerseits die Interaktion mit dem Model übernimmt, also auch validierte Daten an das Model übergeben soll, und andererseits die Daten für das View aufbereiten soll. Deshalb wird vorgeschlagen, das ViewModel in zwei Schichten zu teilen, die diese beiden Aufgaben getrennt voneinander erledigen. Da sich der Funktionsumfang bei dem geplanten Programm in Grenzen hält, ist dies nicht nötig.

### Aufgaben der einzelnen Komponenten

#### View – GUI mit JavaFX 2.0

#### Model – Datenmodell

#### Core – Viewmodel

#### Utilities

##### Serverkommunikation

##### Spielzugberechnung

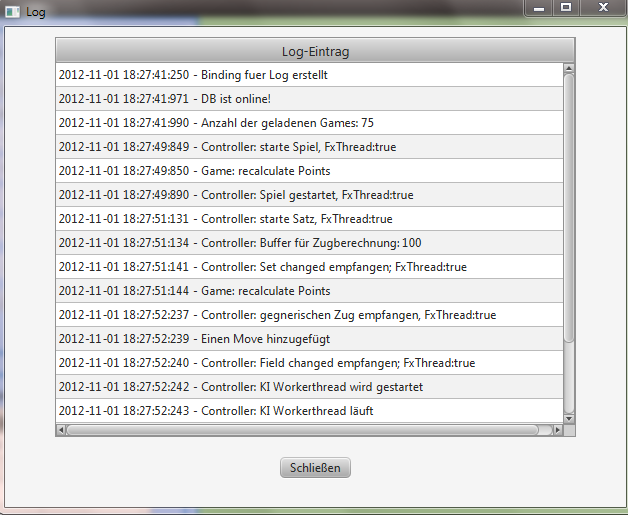
##### Eventhandling

##### Datenbank

Die unterliegende Datenbank ist eine HSQLDB. Die Schnittstelle bildet die Klasse DBConnection. Die Klasse ist als Singleton implementiert, damit über einen internen User auf die Datenbank zugegriffen wird. Es wird dafür der Standard-User verwendet. Dadurch ist die Datenbank generell zugänglich, auch für externe Zugriffe und Änderungen. Die Klasse DBConnection bietet Methoden zum Laden und Speichern von Spielen. Die nach außen sichtbaren Methoden sind mit dem Zusatz synchronized versehen, um Datenbankkonflikte zu vermeiden. Es ist ein Offline-Modus implementiert, damit auch bei Fehlen der Datenbank oder bei gleichzeitigem Zugriff auf die Datenbank über eine andere Schnittstelle das Spielen funktioniert. Das ist nötig, da auch die Datenbank in der JVM läuft.

##### Log

Um interne Zustände des Programmes zu kommunizieren erschien uns eine Hilfsklasse nützlich, die beliebige Log-Einträge ausgeben kann und damit dem Nutzer des Programmes Auskunft über interne Programmparameter geben kann. Sinnvoll ist dies vor allem für das Debuggen des Programmes. Die Ausgabe von Strings ist dabei mit der Ausgabe auf der Konsole vergleichbar, allerdings wollten wir die Konsole nicht nutzen, da das Programm nicht zwangsläufig mit einer Konsole gestartet wird und so diese Ausgabe nicht immer sichtbar wäre. Die Anzeige der Logeinträge kann der Nutzer via Menüpunkt „Hilfe -> Log anzeigen“ auswählen.



Wie man sehen kann wird zu jedem Logeintrag auch die Zeit vorangestellt, zu der dieses Ereignis aufgetreten ist. Dies ermöglicht ein besseres Debugging. Die Logging-Funktion kann im UI ausgeschaltet werden, wenn keine unnötige Performance verschwendet werden soll und sowieso kein Debugging stattfindet.

Die Log-Klasse ist als Singleton implementiert. Die Verwendung für jeden Programmierer gestaltet sich simpel:

Log.getInstance().write("Log-Eintrag"); // Zeitpunkt wird automatisch hinzugefügt

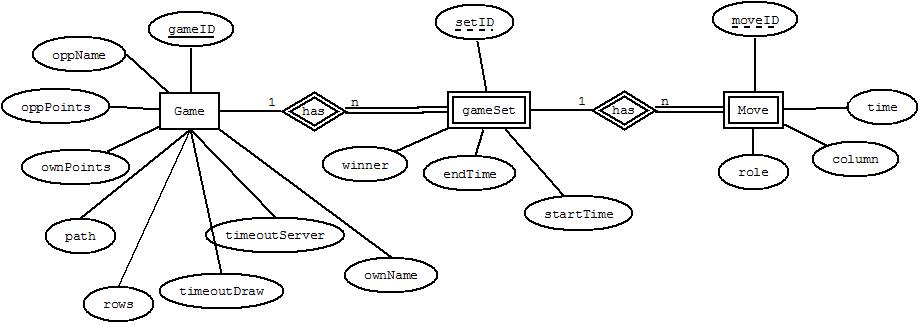
### Tests

## Programmablauf

## Anhang

### Klassendiagramm

### Entity-Relationship Diagramm



### Javadoc

### B.4.4 Sequenzdiagramme

Die wichtigtsten Funktionen sind als Sequenzdiagramm abgebildet im Anhang beigelegt. Dadurch wird die zugrunde liegenden Abhängigkeiten abgebildet. 5 Sequenzdiagramme sind beigelegt:

1. Programm starten: Beim Start des Programms wird in zunächst die Benutzungsoberfläche aufgebaut. Dazu werden die JavaFX-spezifischen Elemente (Stage, FXMLLoader, Scene) geladen. Ebenso werden der MainUIController und der GameController als steuernde Elemente geladen.
2. Spiel starten: Beim Starten eines neuen Spiels werden die in diesem Diagramm abgebildeten Abläufe in Gang gesetzt. Über den GameController wird ein neues Spiel (Game) erzeugt, dieses für Events empfänglich gemacht und in der Datenbank gespeichert.
3. Satz starten: Dieses Diagramm bildet den Fall ab, dass der Benutzer ein neues Spiel startet. Gleich zu Beginn wird exemplarisch gezeigt, wie ein Eintrag ins Log funktioniert, diese Funktion ist aus Gründen der Übersichtlichkeit in den anderen Diagrammen weggelassen worden. Game erzeugt ein neues Satz-Element, in einem neuen Thread wird der CommunicationServer gestartet, worüber das XML-File eingelesen und ausgewertet wird. Bei erfolgreichem Lesen wird über Events der GameController informiert und von dort ein neuer Move hinzugefügt. Ebenso wird die KI in einem neuen Thread gestartet, die dann den neuen Move schreibt.
4. Satz abbrechen: Beim Abbruch eines Satzes geschieht Folgendes: Beim Spielen über einen Server werden der CommunicationServer und der dazugehörige Thread ReadServerFileThread informiert und der Thread beendet sich selbst. Der letzte Zug des Gegners wird noch als Move gespeichert, wenn die KI über den Server spielt und der Gegner gewonnen hat. Der Status wird entsprechend auf beendet gesetzt.
5. Spiel laden: Beim Laden eines Spiels wird die Benutzungsoberfläche angepasst. Der GameController ruft über die DBConnection die Datenbank auf. Zurückgeliefert wird ein Game. Daraufhin werden auch die entsprechenden Sets und Moves abgerufen.

# Projektdurchführung

## Entwicklungsmodell und Testkonzept

### Das Entwicklungsmodell

Bei der Entwicklung wird nach dem Spiral-Modell vorgegangen. Das Modell sieht vor, dass für jeden Entwicklungsschritt ein Zyklus von 4 Phasen durchlaufen wird. Die Phasen gliedern sich folgendermaßen:

1. Ziele: Ziele festlegen, Rahmen definieren
2. Vorgehen: Alternativen bewerten, Risiken bewerten
3. Produkt: Realisierung des Zwischenprodukts
4. Test: Beurteilung der Phasen 1-3, Planung des nächsten Zyklus

Zyklen in diesem Projekt sind:

* Entwicklungszyklus Prototyp: Ziel ist ein vorzeigetauglicher Prototyp
* Entwicklungszyklus Beta-Release: Ziel ist die auslieferungsfähige Beta-Software
* Entwicklungszyklus Release: Ziel ist die Fertigstellung der Software, wobei nach Projektplan in diesem Zyklus hauptsächlich getestet werden soll

Dokumentation der einzelnen Zyklen:

1. Prototyp
   1. Ziele: erfolgreichen Prototypen erstellen; dazu: gemeinsame Überlegungen zur Architektur und daraufhin Arbeit in Kleingruppen an vereinbarten Arbeitspaketen; Erstellung eines Zeitplans
   2. Vorgehen: Einigung auf Zuständigkeiten:

* GUI → Nora
* Datenbank → Henny
* Logik/KI → Johannes
* Spielablauf → Sascha
* Log → Alex
* Kommunikation mit Server → Björn
* Projektorganisation und Dokumentation → Henny und Alex
* Modellierung: alle
  1. Produkt: Erste Entwürfe zu Klassendiagramm und ERD sind entstanden. Info-Sessions über wichtige Themen werden für alle Teammitglieder vorbereitet, Themen sind: Github-Benutzung, Threads, JavaDoc. Programmierung erfolg gemäß den Zuständigkeiten. Testen des Prototypen erfolgt manuell.
  2. Test und Rückblick auf den Zyklus:

Gefundene Modellierungsschwierigkeiten: Eventhandling, Klassendiagramm muss überarbeitet werden, GUI muss angepasst werden

Feedback: sämtliche Dokumente müssen ab sofort als PDF oder JPEG vorliegen; ansonsten soweit in Ordnung

1. Beta-Release
   1. Ziele: Entwicklung der Beta-Version, die bereits 5 Spielzüge spielen kann und dabei mit GUI und Datenbank verknüpft ist. Nebenbei Dokumentation und notwendige Dokumente anlegen. Das Programm soll möglichst weit gebracht sein, so dass im letzten Zyklus das Testen im Fokus liegt.
   2. Vorgehen: Wie bisher hat jeder seinen Aufgabenbereich, da sich diese Aufteilung bewährt hat. Für Absprachen werden die Vorlesungsstunden genutzt.
   3. Produkt: (vgl. Quellcode) Testklassen und –fälle z.T. werden bereits implementiert und entworfen.
   4. Test und Rückblick auf den Zyklus: Beim Spielen auf dem Server treten z.T. einige andere Bedingungen auf als beim Testen auf den eigenen Rechnern; Beim Verpacken in eine ausführbare Jar-Datei sind in Bezug auf die Datenbank einige Punkte zu beachten; Die KI muss noh weiterentwickelt werden; die Stabilität des Programms muss ausgebaut werden.
2. Release:
   1. Ziele: Ziel ist die Abgabe des voll funktionsfähigen Programms, sowie der Dokumentation.
   2. Vorgehen: Weiterhin gibt es die Verantwortlichen für die einzelnen Komponenten. Ein Verantwortlicher für die Dokumentation wurde bestimmt. Außerdem wird ein Testkonzept entwickelt. Auffälligkeiten werden in einer To-Do-Liste mit Zuständigkeit, Erfassungsdatum, Erfasser und Status hinterlegt.
   3. Produkt: Als Produkte können der Agent, die Dokumentation, ebenso wie die Tests angesehen werden.
   4. Test und Rückblick auf den Zyklus: Trotz vorher guter Einhaltung des Zeitplans sind einige unerwartete Überraschungen, insb. Beim Testen mit dem Server, aufgefallen. Da aber ausreichend Zeit eingeplant war für Änderungen, ist es machbar. Automatisierte Tests wären an dieser Stelle in einem größeren Projekt sicher sinnvoll. Da aber die Vorlesungszeit zur Verfügung steht und ausreichend Teammitglieder vorhanden sind, ist die Aufgabe des Testens auch so gut zu bewältigen.

Fazit zum Entwicklungsmodell:

Es war für alle Teammitglieder die erste Erfahrung mit dem Spiralmodell. Positiv ist zu bemerken, dass das Modell recht einfach zu verstehen und umzusetzen ist. Allerdings war es in diesem kleinen Rahmen schwierig, die einzelnen Zyklen voneinander abzugrenzen. Da kaum Vorgaben gemacht werden, wird weiteres Projektmanagement, z.B. ein Zeitplan, benötigt, so dass es beinahe wie nebenläufige Strategien scheint und die Abstimmung beider Mittel zeitaufwendig ist. In einem größeren Projekt wäre dieses Entwicklungsmodell sicher eher zu gebrauchen.

## Javadoc

### Javadoc im Projekt

Javadoc ist eine Anwendung zur Erstellung einer API Dokumentation. Hierbei wird aus den Kommentaren, bei Verwendung einer bestimmten Syntax ein strukturiertes HTML Dokument erzeugt, welches Auskunft über bspw. den Autor und die verwendeten Parameter gibt.

Aufgrund der übersichtlichen Dokumentation welche mit Javadoc erstellt werden kann, haben wir uns zum Einsatz dieses Tools entschieden.

Jeder Projektteilnehmer wird angehalten seinen eigenen Quelltext ordentlich zu kommentieren um eine saubere Umsetzung mit Javadoc zu gewährleisten. Dies hilft den Aufbau und die Funktionen des Projektes für alle nachvollziehbar aufzubauen und Erweiterungen zu erleichtern.

Im Rahmen des WI-Projektes beschränken wir uns auf die folgenden drei Tags:

* @author *Vorname Nachname* – Autorname
* @param *Parametername Beschreibung* – Beschreibt den Parameter einer Methode
* @return *Beschreibung* – Beschreibt den Rückgabewert einer Methode

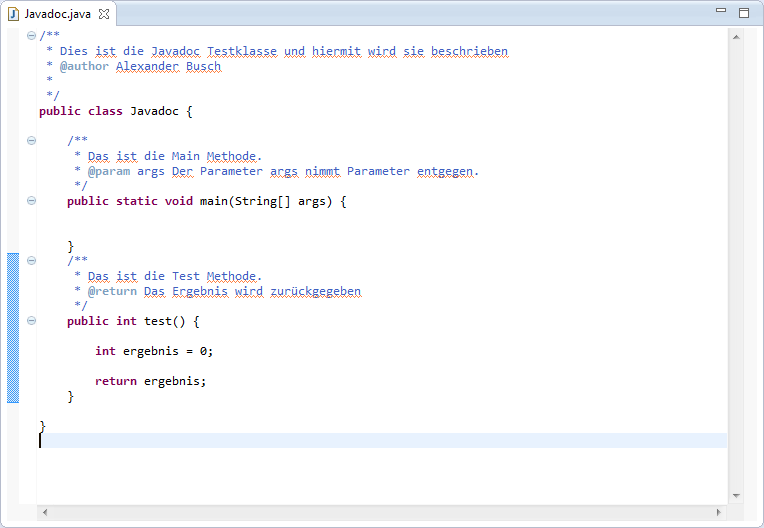
### Javadoc Anleitung

Voraussetzung: Um Javadoc nutzen zu können muss diese Anwendung auf dem Rechner verfügbar sein. Dies bedeutet, dass sich im entsprechenden Java-Verzeichnis eine javadoc.exe befinden muss. Der Pfad lautet in diesem Fall [Pfad zum Java-Verzeichnis]\bin\.

Ist Javadoc nicht vorhanden kann dies über die Installation eines aktuellen Java Development Kits nachgeladen werden. (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>)

Nutzung: Allgemein werden Quelltextkommentare zur Nutzung in Javadoc immer mit /\*\* eingeleitet und mit \*/ beendet. Erstreckt sich die Beschreibung über mehrere Zeilen wird diese Zeile mit einem \* begonnen.

Klassen und Methoden werden erläutert, bevor sie auftreten. Auf der nachfolgenden Grafik sieht man die Beispielklasse Javadoc, welche mit den von uns vereinbarten Tags (author, param, return) versehen wurde.



Erzeugung in Eclipse: Um Javadoc in Eclipse nutzen zu können, muss Eclipse wissen wo die Javadoc Anwendung auf dem Rechner liegt. Diese Einstellung kann direkt beim Erzeugen der Dokumentation durchgeführt werden.

Um die Dokumentation für ein Projekt erzeugen zu lassen klickt man mit der rechten Maustaste auf das entsprechende Projekt und wählt den Menüpunkt „Export“ aus. Dort findet man unter dem Reiter „Java“ den Punkt „Javadoc“. Die Auswahl bestätigt man mit einem Klick auf „Next“.

Hinter dem Feld „Javadoc Command“ klickt man auf „Configure“ und wählt die „javadoc.exe“ im entsprechenden Verzeichnis aus.

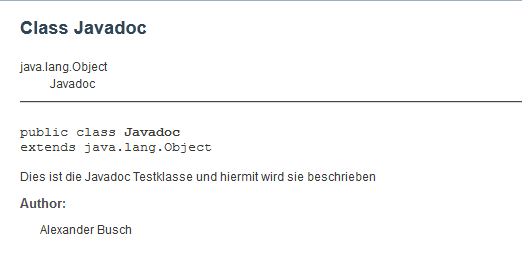
Mit der Einstellung der visibility kann ausgewählt werden welche Methoden und Klassen in der Dokumentation auftauchen sollen.

Die letzte Option ist die Auswahl des Ausgabeverzeichnisses, dieses ist standardmäßig der Ordner „doc“ in der vorher definierten Workbench.

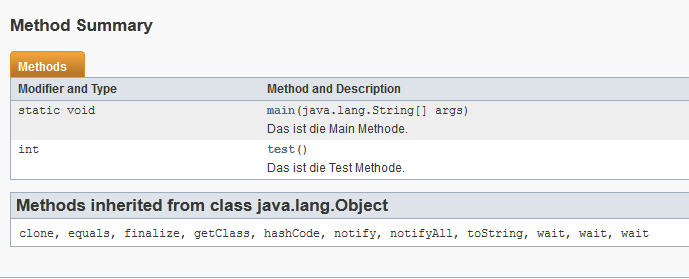
Mit einem Klick auf „Finish“ wird die Dokumentation erzeugt.

Die Dokumentation kann über die „Index.html“ nun aufgerufen werden.

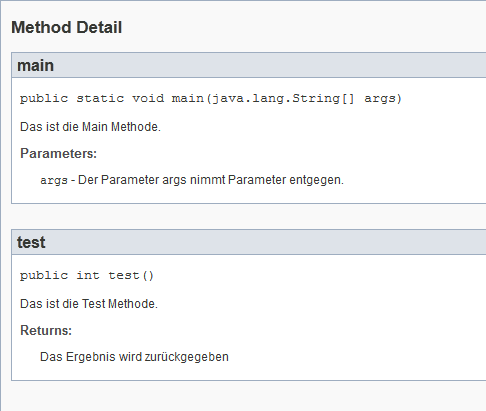
Erläuterung: Die Kommentare, welche wir in der Javadoc Testklasse hinzugefügt haben werden nun übersichtlich Strukturiert dargestellt.



Die Erläuterung der Klasse, sowie des Autors stehen zu Beginn des Dokuments.



Die verwendeten Methoden werden samt Erläuterung angezeigt.



Ebenso wie die Detailansicht der verwendeten Methoden. Hierbei werden auch die Parameter, sowie die Rückgabewerte erläutert.

## Git

Bei Git[[2]](#footnote-2) handelt es sich um ein Programm, welches zur verteilten Versionsverwaltung von Dateien entwickelt wurde. Unter anderem ist der Linux Begründer Linus Torvalds an der Entwicklung und Weiterentwicklung von Git maßgeblich beteiligt.

Die Tatsache, dass viele große Projekte, wie Programmiersprachen oder Betriebssysteme mittlerweile mittels Git verteilt entwickelt werden hat uns zu dem Schritt bewegt, dieses Tool für unser Softwareprojekt einzusetzen. Die Vorteile bestehen in der Versionsverwaltung. Man kann zu jedem Zeitpunkt sehen welcher Projektteilnehmer welche Änderungen vorgenommen hat und ggf. Fehler schnell korrigieren.

Als Hostingplattform für die Daten und deren Verteilung wurde der gitspezifische Hostinganbieter Github[[3]](#footnote-3) ausgewählt. Github bietet seine Dienste konstenfrei an. Die Tatsache, dass Projekte wie PHP, Ruby oder JUnit auf GitHub weiterentwickelt werden förderte die Entscheidung diesen Anbieter auszuwählen.

Um den direkten Datenaustausch aus Eclipse heraus zu ermöglichen wurde das Plugin EGit[[4]](#footnote-4) verwendet. EGit ist das am häufigsten genutzte Plugin zur Nutzung von Git in Eclipse und daher der de-facto Standard.

## Databinding

1. http://www.oracle.com/technetwork/java/javase/downloads/java-se-jre-7-download-432155.html [↑](#footnote-ref-1)
2. http://git-scm.com/ [↑](#footnote-ref-2)
3. https://github.com/ [↑](#footnote-ref-3)
4. http://www.eclipse.org/egit/ [↑](#footnote-ref-4)