

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.....	3
A.1. Grundlagen.....	3
A.2. Erste Schritte.....	3
A.3. Konfiguration.....	4
[A] KI gegen Server.....	4
[B] Spieler gegen KI.....	4
[C] Spielstand laden.....	4
A.4. Das Spiel.....	4
[A] KI gegen Server.....	4
[B] Spieler gegen KI.....	5
[C] Spielstand laden.....	5
A.5. Funktionsmenü.....	5
A.5.1. Spielsteuerung.....	5
A.5.2. Hilfe.....	5
B. Entwicklerdokumentation.....	5
B.1. Ziele und äußerer Rahmen.....	5
B.2. Zielarchitektur.....	6
B.2.1. Model-View-Viewmodel als grundsätzliches Architekturkonzept unseres Programms..	6
B.2.2. Umsetzung MVVM.....	7
B.2.3. Aufgaben der einzelnen Komponenten.....	8
B.2.3.5. Tests.....	<b>Fehler! Textmarke nicht definiert.</b>
B.3. Programmablauf.....	18
B.4. Anhang.....	19
B.4.1. Klassendiagramm.....	19
B.4.2. Entity-Relationship Diagramm.....	20
B.4.3. Javadoc.....	20
B.4.4 Sequenzdiagramme.....	<b>Fehler! Textmarke nicht definiert.</b>

C. Projektdurchführung .....	20
C.1. Entwicklungsmodell und Testkonzept.....	20
C.1.1. Das Entwicklungsmodell.....	20
C.1.2. Testkonzept .....	22
C.2. Javadoc .....	27
3.4.1. Javadoc im Projekt.....	27
3.4.2. Javadoc Anleitung.....	27
C.3. Git.....	30
C.4. Databinding.....	31

## A. Benutzerdokumentation

### A.1. Grundlagen

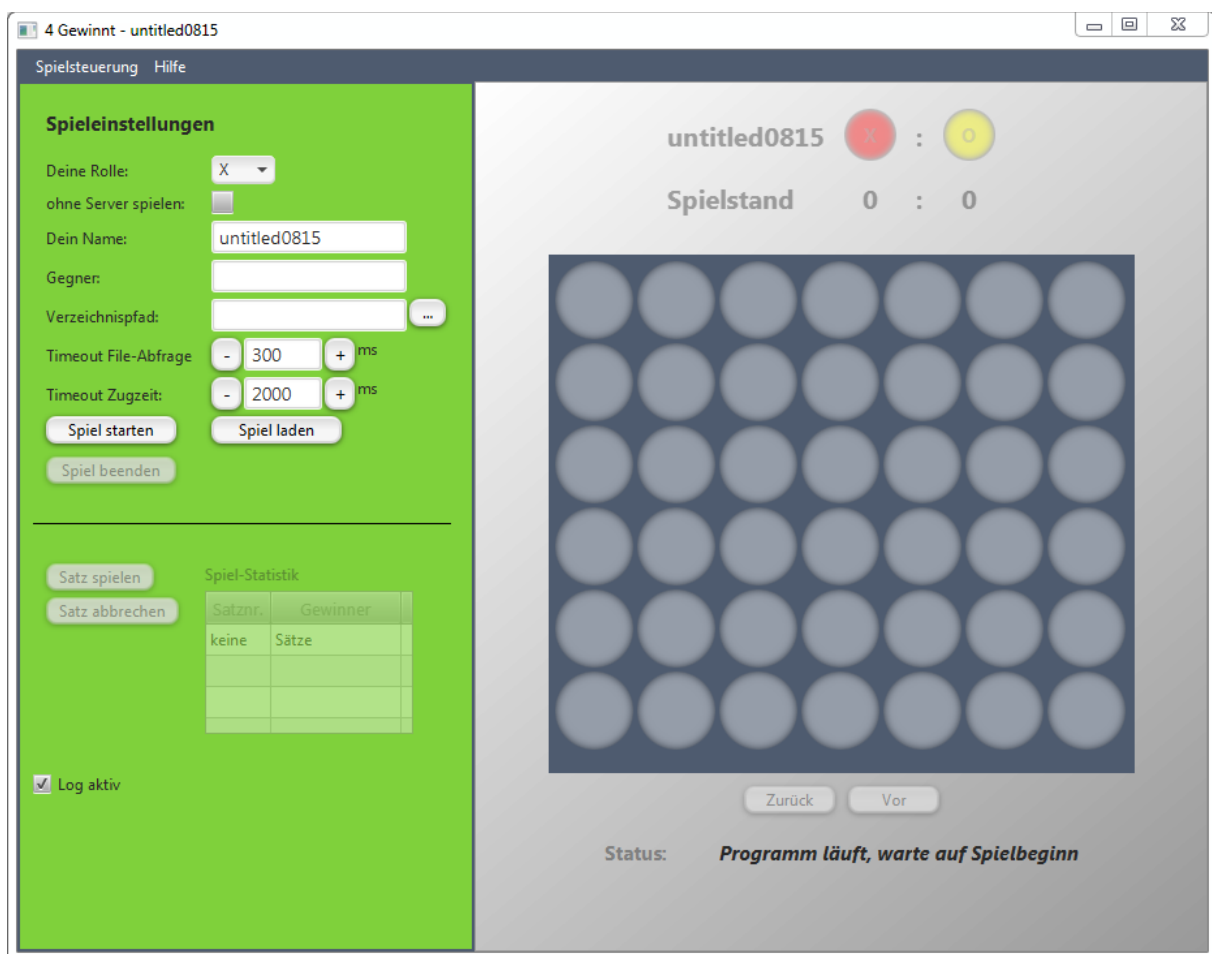
Bei dem vorliegenden Javaprojekt der Gruppe untitled0815 bestehend aus Alexander Busch, Nora Herentrey, Björn List, Johannes Riedel, Henny Selig und Sascha Ulbrich handelt es sich um eine Anwendung, welche es ermöglicht, das bekannte Gesellschaftsspiel Vier Gewinnt in verschiedenen Modi zu spielen.

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.

### A.2. Erste Schritte

Die auf der CD gelieferte Datei 4gewinnt\_untitled0815.zip wird in den Zielordner auf dem Benutzer-Computer extrahiert. Dort steht der Software-Agent als ausführbarer Java-Export (\*.jar) zur Verfügung. Um die Anwendung zu starten, öffnet man diese mit einem Doppelklick auf 4gewinnt\_untitled0815.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.<sup>1</sup>

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



<sup>1</sup> <http://www.oracle.com/technetwork/java/javase/downloads/java-se-jre-7-download-432155.html>

### A.3. 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.

### A.4. 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. Nach Beendigung des Satzes oder Abbruch durch Klicken des Button „Satz abbrechen“ erscheint der Dialog „Gewinner bestätigen“, in welchem mittels Drop-Down Menü der Gewinner bestätigt oder geändert werden kann.
4. Sie können im Anschluss einen weiteren Satz spielen oder das Spiel beenden.

## [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. Nach Beendigung des Satzes oder Abbruch durch Klicken des Button „Satz abbrechen“ erscheint der Dialog „Gewinner bestätigen“, in welchem mittels Drop-Down Menü der Gewinner bestätigt oder geändert werden kann.
4. Sie können im Anschluss einen weiteren Satz spielen oder das Spiel beenden.

## [C] Spielstand laden

1. Je nachdem welche Option sie gewählt haben, bekommen sie nun das Endergebnis des Spiels angezeigt oder können dieses durch die Vor und Zurück Tasten Schrittweise nachverfolgen.

## A.5. Funktionsmenü

### A.5.1. Spielsteuerung

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

### A.5.2. Hilfe

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

## B. Entwicklerdokumentation

### B.1. 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 Dokumentationsfähigkeiten 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

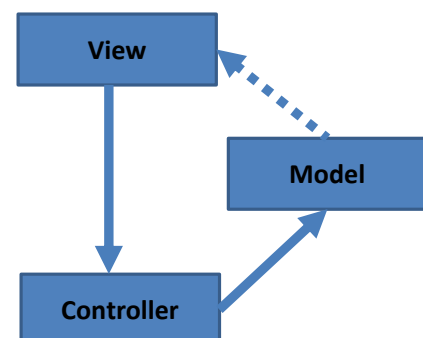
## B.2. Zielarchitektur

### B.2.1. Model-View-Viewmodel als grundsätzliches Architekturkonzept unseres Programms

#### B.2.1.1. 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.

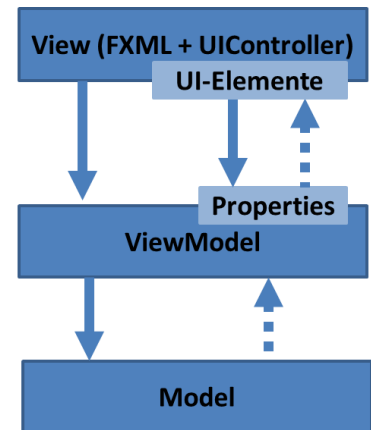


MVC ist allerdings nicht optimal für das zu erstellende Programm geeignet, da keine Schicht zwischen View und Model existiert. Somit ist eine Umsetzung nur nach diesem Entwurfsmuster schwierig, da die Benutzeroberfläche nicht direkt mit dem Datenmodell arbeiten, sondern ein extra UI-Datenmodell erstellt 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.

#### B.2.1.2. Model-View-Viewmodel als genutzte Weiterentwicklung

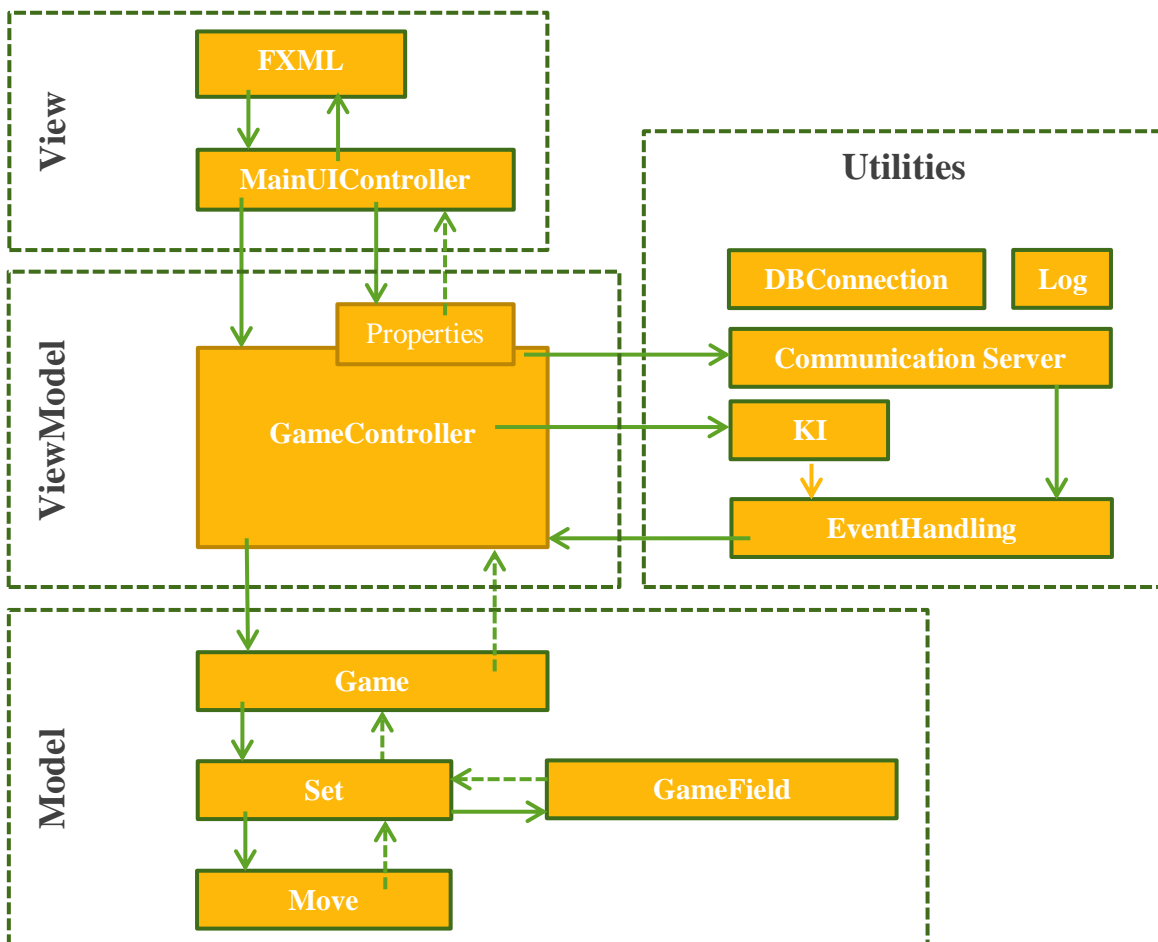
Als passende Alternative hat sich das Model-View-ViewModel (MVVM)-Entwurfsmuster angeboten, das eine bessere Trennung von UI-Design und UI-Logik ermöglicht.

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.



### B.2.2. Umsetzung MVVM

Für das zu erstellende Programm wird folgende Architektur auf Basis von MVVM gewählt:





#### **B.2.2.1. View**

Der View nach MVVM wird durch die FXML „MainUI.fxml“ und den „MainUIController“ abgebildet. In der FXML ist rein deklarativ der Aufbau des UIs beschrieben, während der UIController die Logik des UIs beinhaltet. Beide Elemente sind von daher eng mit einander verbunden.

#### **B.2.2.2. ViewModel**

Das ViewModel nach MVVM wird durch die Klasse GameController umgesetzt. Der GameController stellt Methoden und Properties für den View zur Verfügung. Der View verbindet sich mittels Data Binding mit den Properties wodurch Änderungen der Properties im GameController und View synchronisiert werden. Der GameController kontrolliert das Model und steuert den Programmablauf.

#### **B.2.2.3. Model**

Das Model wird durch die Klassen Game, Set, Move und GameField gebildet. Das ViewModel greift primär über Game auf das gesamte Model zu. Ein Game beinhaltet mehrere Sets. Ein Set beinhaltet mehrere Moves und ein GameField.

#### **B.2.2.4. Utilities**

Zur Unterstützung des ViewModel werden unter anderem folgende Klassen implementiert:

- DBConnection – zum Verwalten der Datenbankverbindung
- Log – zur Erstellung eines zentralen Logs
- CommunicationServer – zum Verwalten der Serververbindung
- KI – zur Berechnung von Zügen und Gewinnerkennung
- EventHandling – zur Übermittlung von asynchronen Nachrichten an das ViewModel

### **B.2.3. Aufgaben der einzelnen Komponenten**

#### **B.2.3.1. View – GUI mit JavaFX 2.0**

JavaFX basiert auf einer Skript-Sprache, und ist für das Java-Umfeld entwickelt worden. Mit JavaFX kann insbesondere die Darstellung von Oberflächen im Vergleich zu Swing einfacher programmiert werden. JavaFX läuft im Browser, außerhalb des Browser, auf verschiedenen Geräten und ist flexibel einsetzbar. Durch die Einbindung in Java stehen viele Bibliotheken zur Verwendung bereit. Seit JavaFX 2.0 kann FXML, eine XML-Sprache, zum Definieren der Oberflächen-Elemente benutzt werden. Aus den bereits genannten Gründen und dem gegebenen Funktionalitätsumfang ist die Benutzungsoberfläche mit JavaFX und FXML umgesetzt.

#### **B.2.3.2. Model – Datenmodell**

Game stellt das hierarchisch höchste Element im Model dar. Es regelt den Zugriff auf das komplette Model. Game beinhaltet eine beliebige Anzahl Sets, die über Game abgerufen werden können. Außerdem wird der Punktestand anhand der vorhandenen, beendeten Sets bestimmt.

Ein Set beinhaltet alle Züge eines Satzes in Form von beliebig vielen Move Objekten und ein GameField in dem das aktuelle Spielfeld in Form eines zweidimensionalen Arrays gespeichert wird. Die Koordinate 0;0 entspricht dabei dem Stein in der ersten Spalte von Links, ganz unten.

Die Elemente Game, Set und Move implementieren save-Methoden um die Speicherung des Datenmodells zu ermöglichen. Die zu speichernden Objekte werden mittels DBConnection persistiert.

#### **B.2.3.3. Core – Viewmodel**

Der GameController setzt das ViewModel um und koordiniert damit auch den Programmablauf. So wird die KI vom GameController zur Berechnung des nächsten Zuges aufgerufen oder dem CommunicationServer mitgeteilt auf dem Server zu lesen, oder eben dies zu beenden. Weitere Aufgaben können in folgende Aufgabenbereiche unterteilt werden:

##### **1. Bereitstellung von Methoden und Properties für den View**

Der GameController bietet eine Vielzahl von Properties an, an die sich das View binden kann. Durch diese Properties werden alle UI-Relevanten Informationen zur Verfügung gestellt. Diese Informationen basieren größtenteils auf Informationen aus dem Model die für das UI aufbereitet werden, allerdings sind diese Daten nicht designspezifisch. Außerdem werden über Properties auch Daten von dem View an den GameController übergeben, ein so genanntes bidirektionales Binding.

Des Weiteren stellt der GameController auch Methoden bereit über die der View bestimmte Aktionen auslösen kann. Beispielsweise der Start eines neuen Satzes. Diese Aktionen führen meist zu Aktionen im Datenmodell.

##### **2. Behandlung von Änderungen im Datenmodell**

Der GameController kontrolliert das Modell und reagiert auf Änderungen im Datenmodell in dem bei Änderungen des Gewinners, der Sätze, des Spielfeldes oder des Punktestandes die Properties entsprechend angepasst werden um den aktuellen Stand wieder zu spiegeln.

##### **3. Behandlung von Events**

Um die Koordination mit weiteren Elementen zur Bewerkstelligen implementiert der GameController das GameEventListener Interface um alle GameEvents empfangen zu können. Diese Events werden vor allem von CommunicationServer genutzt um über neue Ereignisse von Serverseite zu informieren und von der KI um über eine vorhandene Gewinnsituation zu informieren.

##### **4. Laden eines Spiels**

Für die Funktionalität ein Spiel zu laden, lädt der GameController über die DBConnection das ausgewählte Game, seine Sets und die dazugehörigen Moves. Je nach Auswahl wird das Datenmodell sofort zusammengesetzt oder jeder einzelne Zug nach Nutzerwunsch visualisiert.

#### **B.2.3.4. Utilities**

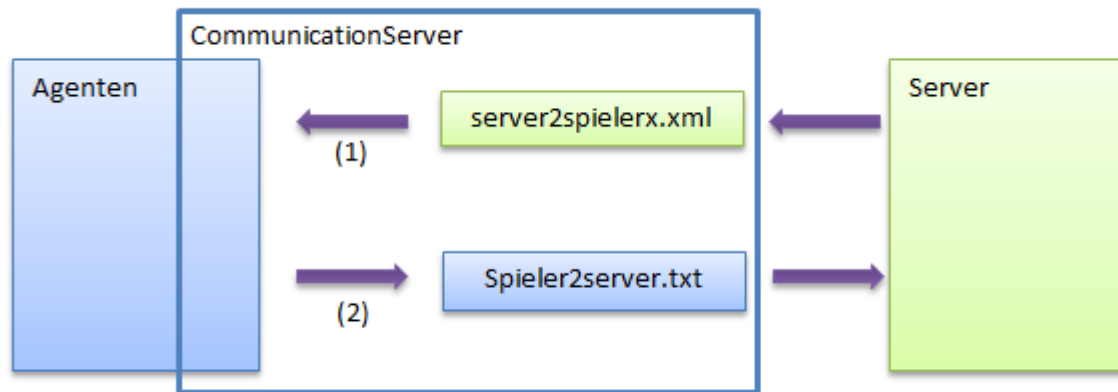
##### **B.2.3.4.1. Serverkommunikation**

Dieses Paket enthält die notwendigen Bestandteile zu Kommunikation mit dem Server und somit zum Spielbetrieb über das Netzwerk.

##### **B.2.3.4.1.1. Aufgaben**

Dieses Paket erfüllt zwei Hauptaufgaben, die in der unten stehenden Abbildung veranschaulicht werden. Links ist die Agentensoftware zu sehen und rechts der Server. In der Mitte befinden sich die Dateien zum Austausch von Informationen zwischen Agent und Server. Dabei beschreibt die Farbe

der Datei, von wem diese geschrieben wird.



#### *(1) Überwachung der Serverdatei für den Agenten*

Der Agent überwacht in einem auf der GUI einzustellenden Frequenz das angegebene Verzeichnis, ob eine neue Datei vom Server geschrieben wurde. Dabei ist zu beachten, dass nur neue Dateien gelesen werden und keine bereits gelesenen Dateien nochmals eingelesen werden.

Wenn eine neue Datei erkannt wird, startet das Parsen dieser XML Datei. Das Ergebnis ist eine Objekt der Klasse `ServerMessage`, dass alle Informationen der Serverdatei enthält. Im Anschluss werden diese Informationen überprüft und entsprechende Events zur weiteren Verarbeitung werden erzeugt.

Liste der möglichen Events:

- `OppMove`: Zug des Gegners inklusive der jeweiligen Spalte (erster Spielzug mit Spaltenwert - 1 wird ebenfalls als `OppMove` abgebildet)
- `WinnerSet`: Ein Gewinner ist vom Server gesetzt worden. Zusätzlich wird die Rolle des Gewinners mit übermittelt.
- `EndSet`: Der Satz wurde vom Server beendet.

Diese Events werden erzeugt und über den `EventDispatcher` (siehe Kapitel Eventhandling) weitergeleitet.

#### *(2) Zurückschreiben der Züge des eigenen Agenten an den Server*

Die zweite Aufgabe besteht in der Antwort des Agenten an den Server. Dazu wird eine Textdatei mit der Spaltennummern des Agentenzuges in das Serververzeichnis durch den Agenten geschrieben. Dabei wird nochmals überprüft, ob der zu schreibende Wert gültig ist.

#### **B.2.3.4.1.2. Performanceoptimierung**

Ein wichtiger Bestandteil der Überwachung ist, dass diese möglichst parallel abläuft und dem Programm die Möglichkeit gibt, Berechnungen und andere Funktionen gleichzeitig auszuführen. Dazu wird die erste Funktion, das Überwachen der Serverdatei, durch einen eigenen Thread realisiert. Die dazu notwendige Implementierung wurde in der Klasse `ReadServerFileThread` durchgeführt. Der Start ist auch im Sequenzdiagramm „Satz starten“ zu sehen.

### B.2.3.4.1.3. Bestandteile

Für die Aufgaben der Serverkommunikation kommen folgende Klassen zum Einsatz:

#### 1. CommunicationServer

Singleton, der den anderen Paketen als zentrale Schnittstelle dient. Über das Objekt wird die Serverdatei Überwachung gestartet und kann jederzeit wieder beendet werden. Die Klasse leitet die Informationen der Serverdatei über Events an das Hauptprogramm weiter. Des Weiteren verwaltet das Objekt die Pfade zu den Dateien und die Dateinamen, die entsprechend der eigenen Rolle definiert werden. Außerdem wird die Agenten Datei an den Server durch diese Klasse geschrieben,

Eine weitere Funktion ist das Testen der Schreibgeschwindigkeit. Damit soll sichergestellt werden, dass die Timeoutzeiten nicht durch einen schlechten Netzwerkzugriff überschritten werden.

#### 2. ReadServerFileThread

Diese Klasse dient ausschließlich dazu, die Überwachung der Serverdatei als eigenen Thread zu erzeugen um eine parallele Abarbeitung zu ermöglichen.

#### 3. XMLParser

Das Parsen und auswerten des Serverfiles für die Bereitstellung der Informationen erfolgt in dieser Klasse. Sie ist ebenfalls als Singleton realisiert. Sie liest das Serverfile aus, parst das XML und erstellt, soweit die XML wohl geformt ist, eine ServerMessage Instanz.

#### 4. ServerMessage

Dieses Objekt bietet die Datenpersistenz für die Nachrichten vom Server an den Agenten. Für jede Servernachricht wird eine Instanz erzeugt.

### B.2.3.4.2. Spielzugberechnung

#### B.2.3.4.2.1. Alpha-Beta-Suche als allgemeines Prinzip

Zur Berechnung des nächsten Spielzuges verwendet unsere KI einen Algorithmus zur Alpha-Beta-Suche. Da diese eine Weiterentwicklung des Minimax-Algorithmus ist, soll im Folgenden erst dieser beschrieben werden. Der **Minimax-Algorithmus** ist ein Algorithmus zur Bestimmung von Spielzügen in Zwei-Personen-Spielen. Grundgedanke ist, dass davon ausgegangen wird, dass jeder Spieler den Zug auswählt, der ihm am meisten Vorzüge in der jeweiligen Situation verspricht. Dabei kalkuliert er mögliche Erwidern des Gegners ein. Es wird davon ausgegangen, dass dieser Gegner ebenfalls optimal spielt. Kalkuliert die KI jede mögliche Erwidern des Gegners und die darauf folgende eigene Erwidern etc. bis zum Ende des Spiels ein, kann sie optimal spielen und ist unabhängig von der Strategie des Gegners.

Zentraler Baustein des Minimax-Algorithmus ist die **Bewertungsfunktion**, die jeder Stellung zuordnet, wie gut sie für den eigenen Spieler ist. Ein Spielfeld, bei dem das eigene Team gewinnt, bekommt einen sehr hohen Wert zugeordnet(z.B. +100), wohingegen ein Spielfeld bei dem der Gegner

gewonnen hat einen sehr niedrigen Wert zugeordnet bekommt (-100). Bietet das gegenwärtige Spielfeld für keinen der beiden Spieler einen Vorzug, sollte eine ideale Bewertungsfunktion den Wert 0 zuordnen. Der Spieler wird deshalb versuchen, einen Spielzug durchzuführen, der ein möglichst gut bewertetes Spielfeld für ihn verspricht (**Maximierungsstrategie**). Gleichfalls wird der Gegner diejenigen Spielzüge durchführen, die ihm eine möglichst niedrige Bewertung (**Minimierungsstrategie**) sichern.

Grundsätzlich kalkulieren beide Spieler die Erwiderungen des Gegners mit ein, sodass sich ein **Spielbaum** aufbaut (siehe Abbildung)

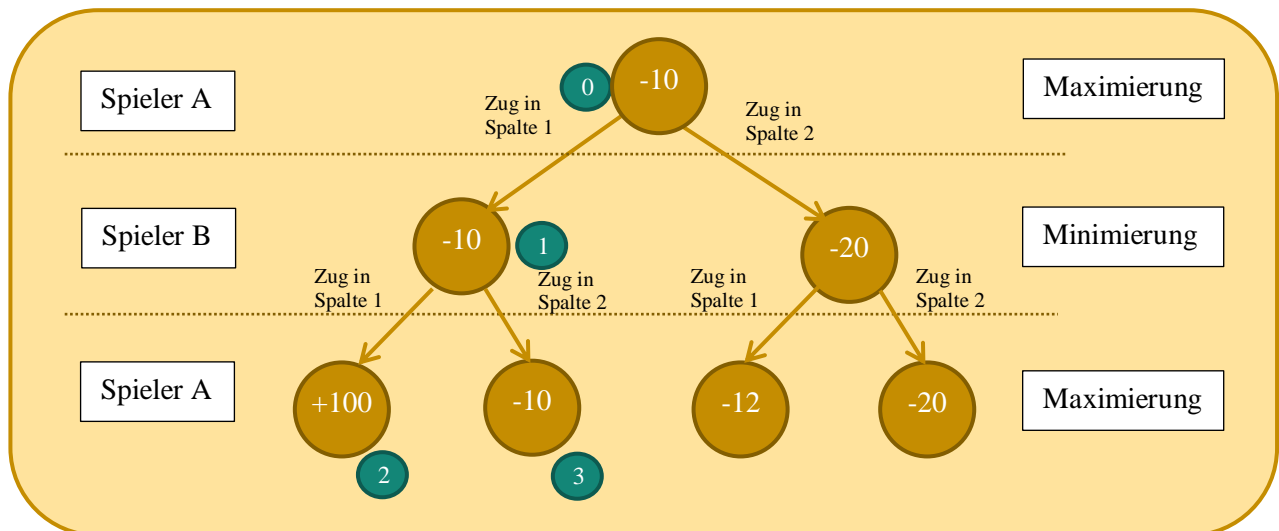


Abbildung: Minimax-Spielbaum der Tiefe 3

Der Aufbau dieses Spielbaumes soll kurz erläutert werden. Es wird davon ausgegangen, dass Spieler A am Zug ist und in dem Spiel grundsätzlich die Optionen „Spalte 1“ und „Spalte 2“ zur Verfügung stehen (weil beispielsweise alle anderen Spalten schon voll sind). Der Spieler probiert den Spielzug in Spalte 1 aus (Knoten 1). Er muss nun herausfinden, was der Gegner erwidern würde, um einzuschätzen, wie gut dieser Spielzug ist. Der Gegner (Spieler B) versucht seinerseits mögliche Spielzüge von A zu kalkulieren. Dabei probiert er zunächst die erneute Erwiderung auf Spalte 1 und gelangt zum Knoten 2. Da die Suchtiefe erreicht ist, kommt die Bewertungsfunktion zum Zug, die ermittelt, dass bei dieser Spielzug-Folge Spieler A gewonnen hätte (da die Bewertungsfunktion +100 zurückgibt). Spieler B prüft anschließend als möglichen eigenen Spielzug die Spalte 2 und gelangt zum Knoten 3. Auch hier wird die Bewertungsfunktion aufgerufen, die diesem Spielfeld den Wert -10 zuordnet. Dieser Wert ist wesentlich besser als der in Knoten 2 ermittelte (Ziel für B ist die Minimierung), sodass sich der Gegner in diesem Fall für den Zug in Spalte 2 entscheiden würde. Da jetzt beide Zugoptionen durchkalkuliert wurden kann dem Knoten 1 ein Wert von -10 zugeordnet werden, der dem Minimum der beiden auf der unteren Ebene berechneten Werte (+100, -10) entspricht. Spieler A weiß jetzt also, dass der Zug in Spalte 0 eine Bewertung von -10 zur Folge hätte (der Gegner also etwas im Vorteil wäre, da die Bewertung kleiner als 0 ist). Anschließend kalkuliert Spieler A für seinen Knoten 0 den Zug in Spalte 2 und mögliche Erwiderungen des Gegners analog zum eben beschriebenen Verfahren durch. Da dem Knoten eine Wertigkeit von -20 zugeordnet wird, entscheidet er sich gemäß seiner Maximierungsstrategie (er will die höchstmögliche Spielfeld-Bewertung) für den Zug in Spalte 1.

Dieser Minimax-Algorithmus kann durch die Alpha-Beta-Suche verbessert werden. Auffallend ist, dass bei oben gegebenem Spielbaum einige Zweige gar nicht erst durchsucht werden müssen, weil sie

sowieso keinen Einfluss auf das Spielergebnis haben würden. Dies soll im Folgenden anhand des obigen Spielbaums erklärt werden.

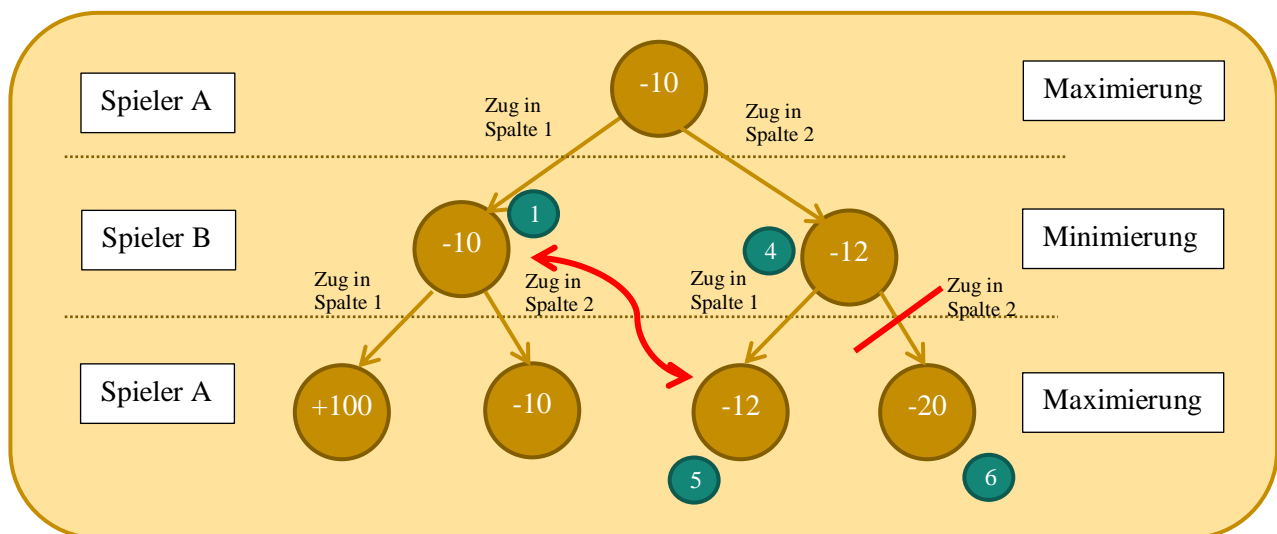


Abbildung: Mögliches Abschneiden im Spielbaum („Cutoff“)

Wir betrachten die Situation nachdem der Algorithmus wie oben beschrieben zurück zum Ausgangsknoten gegangen ist und jetzt den Zug in Spalte 2 analysiert (rechter Teilbaum). In Knoten 4 sucht er nach möglichen Erwidern des Gegners, der seinerseits eine Minimierung des Spielfeldes hinsichtlich der Bewertungsfunktion versucht. Dem Algorithmus ist das Ergebnis von Knoten 1 bekannt, also ein möglicher Spielzug für A, bei dem er im schlechtesten Fall mit dem Wert -10 abschneidet. Dieser Wert stellt somit eine untere Schranke dar: Findet B in einem anderen Ast eine Erwidern, die noch schlechter für A bzw. besser für B (also kleiner als -10) ist, würde sich A auf keinen Fall für diesen Spielzug entscheiden. Genau dies passiert im Knoten 5: Der Gegner könnte so erwidern, dass das Spielfeld mit -12 bewertet wird, also schlechter als in dem Spieler A bekannten Spielzug im linken Teilbaum. Das heißt der Knoten 4 wird mit maximal -12 bewertet, da Spieler B immer den minimalen Ast auswählt. Laut Minimax-Algorithmus würde der Algorithmus jetzt noch Knoten 6 untersuchen, um die optimale Erwidern für B im Knoten 4 zu finden. Dies ist jedoch für das Ergebnis, nämlich den besten Spielzug für A, unerheblich: Der Wert von 4 kann sich nur noch von -12 aus weiter verringern (im Beispiel auf -20) – er bleibt damit aber immer unter dem Wert des Knoten 1. Das heißt sobald Knoten 5 besucht wurde ist bekannt, dass Spieler A den Knoten 4 niemals aufsuchen wird, da er bereits im Knoten 1 einen besseren Spielzug kennt. Das heißt, dass Knoten 6 nicht besucht werden muss, er kann abgeschnitten werden, was auch als „Cutoff“ bezeichnet wird. Die Schranken, zwischen denen der Zielwert liegen muss (also beispielsweise -10 als untere Schranke) werden Alpha und Beta genannt. Dies begründet auch den Namen des Algorithmus: Alpha-Beta-Suche.

#### B.2.3.4.2.2. Unsere Bewertungsfunktion

Zentral für die erfolgreiche Implementation der Alpha-Beta-Suche ist die Bewertungsfunktion, die jedem Spielfeld einen Wert zuordnet, der angibt, wie gut die Steine für den Spieler liegen. Ein wichtiger Teilaspekt ist die **Siegmustererkennung**: erkennt die Bewertungsfunktion vier zusammenhängende Steine des eines Spielers, muss sie einen sehr hohen Wert zurückliefern (z.B. +100), gewinnt der Gegenspieler, einen sehr niedrigen (-100). Um ein möglichst effizientes Abschneiden im Alpha-Beta-Algorithmus zu ermöglichen (viele Cutoffs, die Zeiteinsparungen

bedeuten) muss es hingegen auch Zwischenstufen geben, die angeben, wie nah ein Spieler am Gewinn ist. Dazu stellt sich die Frage, welche Kombinationen für einen Spieler vorteilhaft sind.

Dazu sei der Begriff der „Falle“ erklärt: Eine Falle ist ein Verbund von Steinen desselben Spielers, die zu vier zusammenhängenden Steinen vervollständigt werden kann. Zwei solcher Fallen sind in folgender Abbildung gegeben:

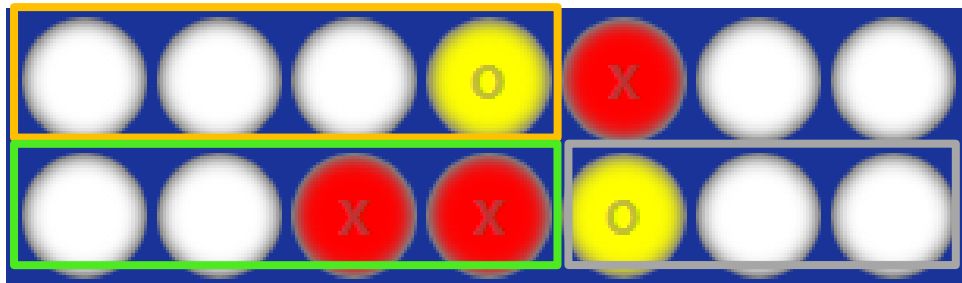


Abbildung: Mögliche horizontale Fallen (grün und gelb umrandet)

Die grüne Umrahmung zeigt dabei eine 2er-Falle für rot – vier Steine in einer Reihe sind möglich. Darüber befindet sich eine Falle der Größe 1 für Gelb. Der graue Kasten zeigt dabei **keine** Falle: Eine Erweiterung auf vier zusammenhängende Steine ist nicht mehr möglich. Natürlich sind größere Fallen, also Fallen bei denen möglichst schon drei Steine zusammenhängen, besser als kleinere Fallen. Wir haben uns entschieden jede Falle wie folgt zu bewerten:

Fallen-Größe	Punkte
1	1
2	4
3	9

Diese Punkte werden jeweils zur Bewertung hinzugezählt oder abgezogen. Eigene Fallen führen zu einer höheren Punktzahl (Punkte werden hinzugerechnet), Fallen des Gegners führen zu einer Verminderung der Punktzahl um die jeweilige Zahl. Die Zählweise soll abschließend an einem gefüllten Feld gezeigt werden:

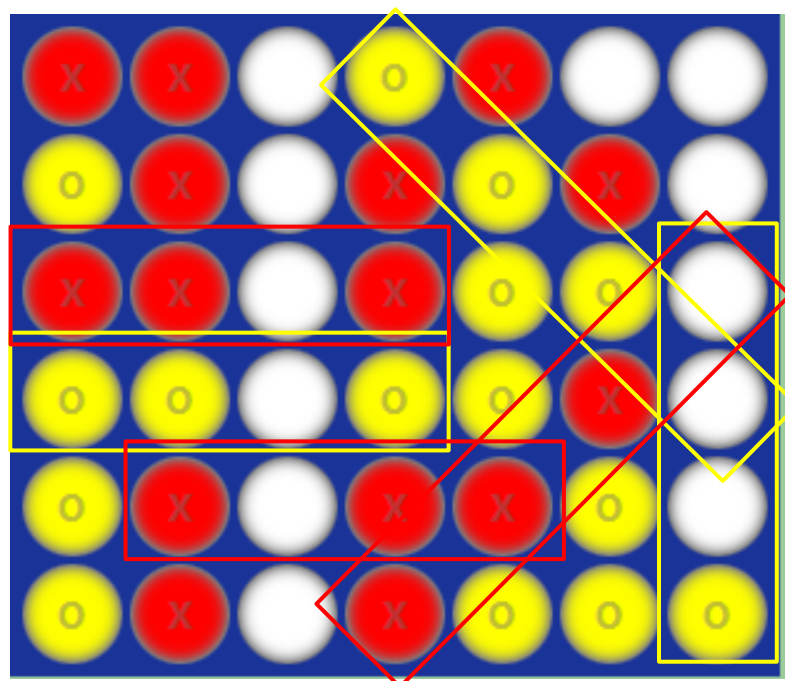




Abbildung: Spielfeld mit markierten Fallen

Punktzahl-Ermittlung (wir sind der gelbe Spieler):

Fälle	Punktzahl
<b>Gelb</b>	
Horizontal Größe 3	+9
Diagonal Größe 3	+9
Vertikal Größe 1	+1
<b>Rot</b>	
Horizontal Größe 3	-9
Horizontal Größe 3	-9
Diagonal Größe 3	-9
	<b>Summe -8</b>

Dieses Spielfeld ist also vorteilhaft für den roten Spieler.

#### B.2.3.4.2.3. Programmtechnische Umsetzung

Alle zur KI benötigten Klassen befinden sich im Subpackage „KI“ innerhalb von Utilities. Die Hauptklasse, die die Schnittstelle nach außen bildet, ist die **KI.java** mit ihrer Methode **calculateNextMove**. Diese Methode startet einen eigenen Thread, der sich mit der Spielzugsuche mittels Alpha-Beta-Suche beschäftigt (KIThread.java). Dieses Konstrukt dient der Überwachung des Timeouts: Von außen wurde der KI im Konstruktor übergeben, wie viel Zeit für die Spielzugberechnung zur Verfügung steht. Diese Zeit läuft der eigene KI-Thread und schreibt währenddessen die Zwischenergebnisse in eine Variable. Ist die Berechnungszeit abgelaufen, beendet die calculateNextMove-Methode innerhalb der Hauptklasse den Berechnungsthread und gibt das aktuelle Ergebnis der Berechnung an den Aufrufer der calculateNextMove-Methode zurück.

Die Alpha-Beta-Suche ist innerhalb der Java-Klasse **KIThread.java** implementiert. Diesem Thread werden im Konstruktor die aktuellen Parameter zum Spielfeld mitgegeben. Den Rahmen bildet die run-Methode des Threads, welche die Alpha-Beta-Suche startet. Die Methoden Min und Max dienen der entsprechenden Spielzugkalkulation für Spieler A und B. Die Methode Rating dient der Berechnung der Spielfeld-Bewertung wie oben beschrieben.

#### B.2.3.4.3. Eventhandling

Um eine Parallelisierung in unserem Programm zu ermöglichen, ist es notwendig eine asynchrone Kommunikation innerhalb zu ermöglichen. Diese Aufgabe erfüllen Events und die zugehörigen Verwaltungsklassen.

#### Bestandteile

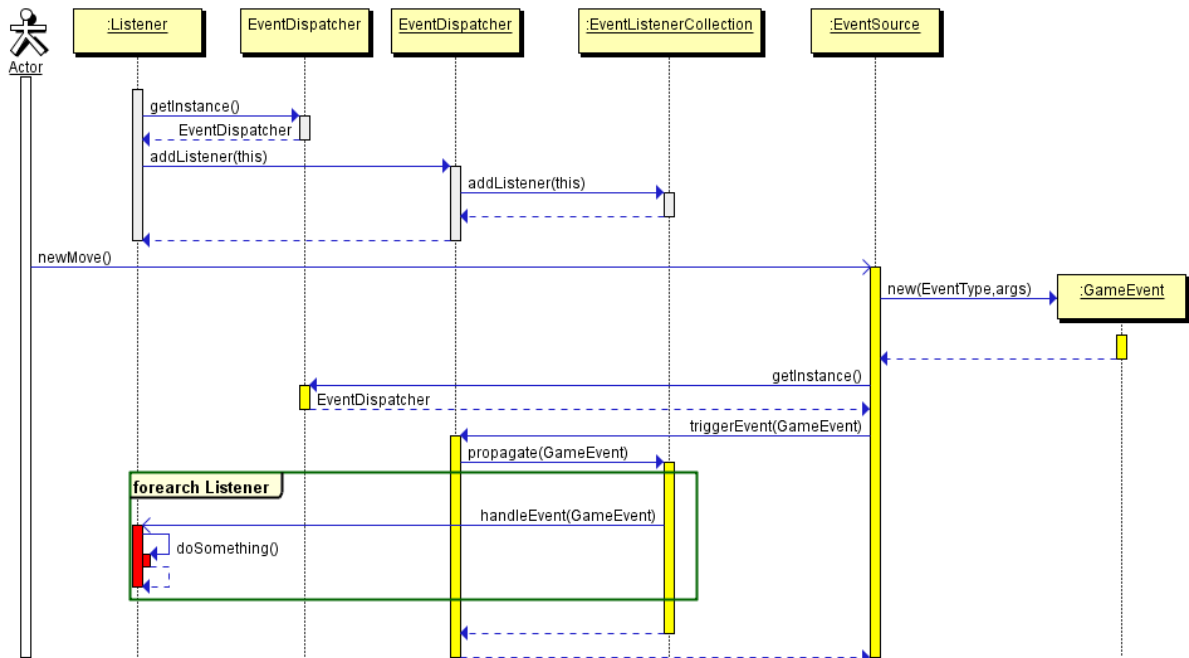
Folgende Bestandteile sind für das Event Handling implementiert worden.

1. Listener – Event Empfänger
2. Event Dispatcher – Zentrale Verwaltung
3. Event Listener Collection – Verwaltung der Empfänger
4. Event Source – Erzeuger eines Events
5. Event – Das Event(die asynchrone Nachricht)



## Ablauf

Im Folgenden soll das Zusammenspiel dieser Komponenten erläutert werden. In dem abgebildeten Sequenzdiagramm ist das genaue Vorgehen zu sehen.



Die grauen Aktivitätsschritte beschreiben die Vorbereitungen zum Empfang eines Events. Der Listener, also derjenige, der ein Event empfangen möchte, muss sich als Listener zentral registrieren. Dazu kontaktiert er den EventDispatcher. Dieser Singleton übernimmt die zentrale Verwaltung für die Events. Über die Methode `addListener()` kann sich ein Listener registrieren. Im EventDispatcher wird dieser Listener dann in eine `EventListenerCollection` aufgenommen. Diese Collection wird für jeden Event Typ erzeugt. Da wir nur ein Event Typ, nämlich das Game Event mit diversen Eigenschaften benötigen, wird nur eine Collection erstellt.

Mit den gelben Aktivitäten wird nun das Auslösen eines Events beschrieben. Aus der EventSource, dem Sender, wird eine neue Instanz von `GameEvent` erzeugt. Nun muss diese Event asynchron versendet werden. Dazu wird der Dispatcher über `triggerEvent()` benachrichtigt. Dieser ruft nun die Collection mit der Methode `propagate()`. In dieser Methode wird nun für jeden Listener die Methode `handleEvent()` aufgerufen. Diese Methode ist als Interface definiert und jeder Listener muss diese Methode implementieren.

Bis jetzt ist dieser Aufruf jedoch synchron. Die roten Aktivitäten zeigen nun einen asynchronen Aufruf. Für jeden Listener wird die `handleEvent()` Methode in einem neuen Thread gestartet und läuft somit unabhängig von der `EventListenerCollection` weiter.

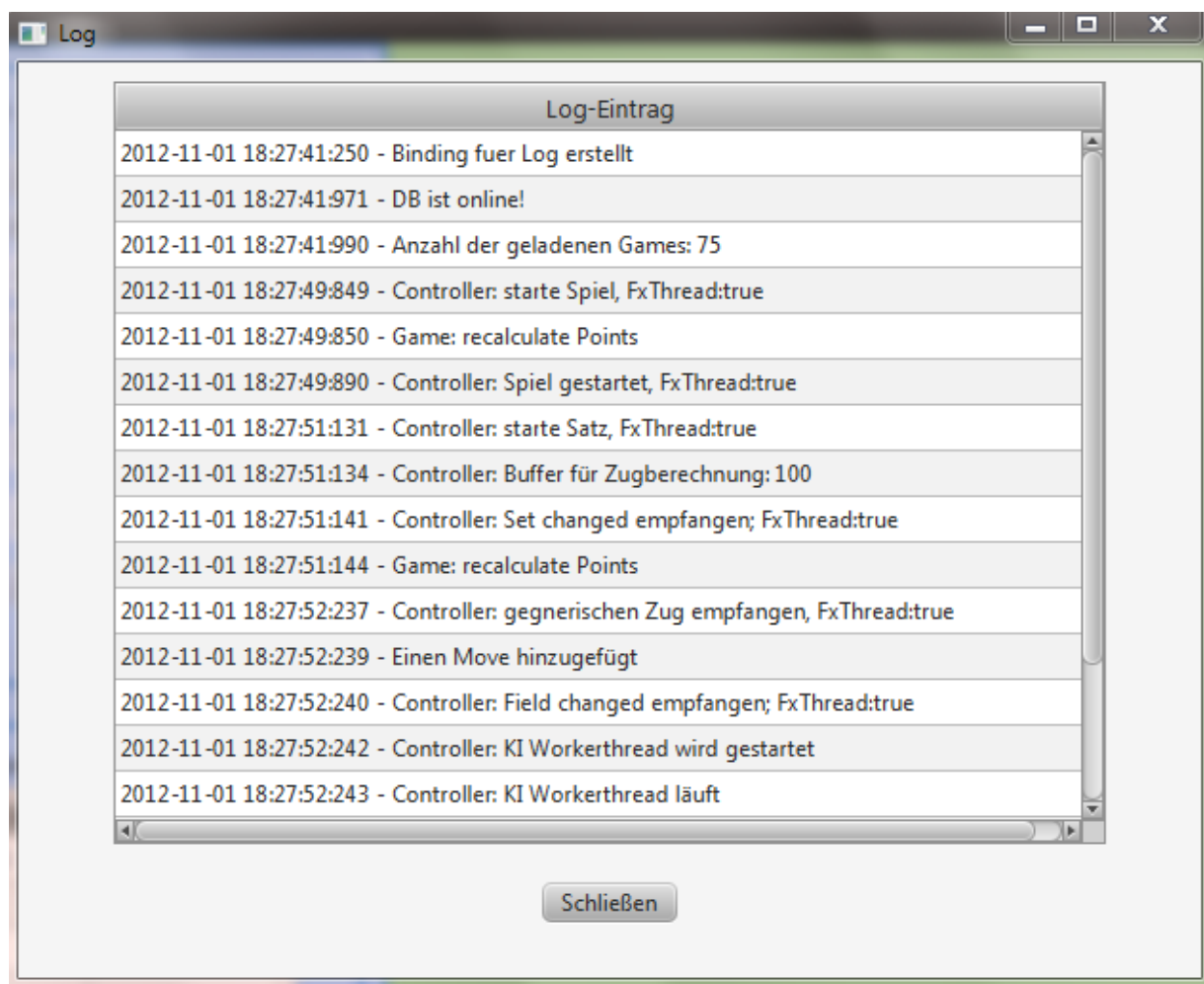
Mit dieser Architektur kann nun parallel im Programm gearbeitet werden und die heute verwendeten Multicoresysteme deutlich besser ausgenutzt werden.

#### B.2.3.4.4. 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.

#### B.2.3.4.5. 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
```

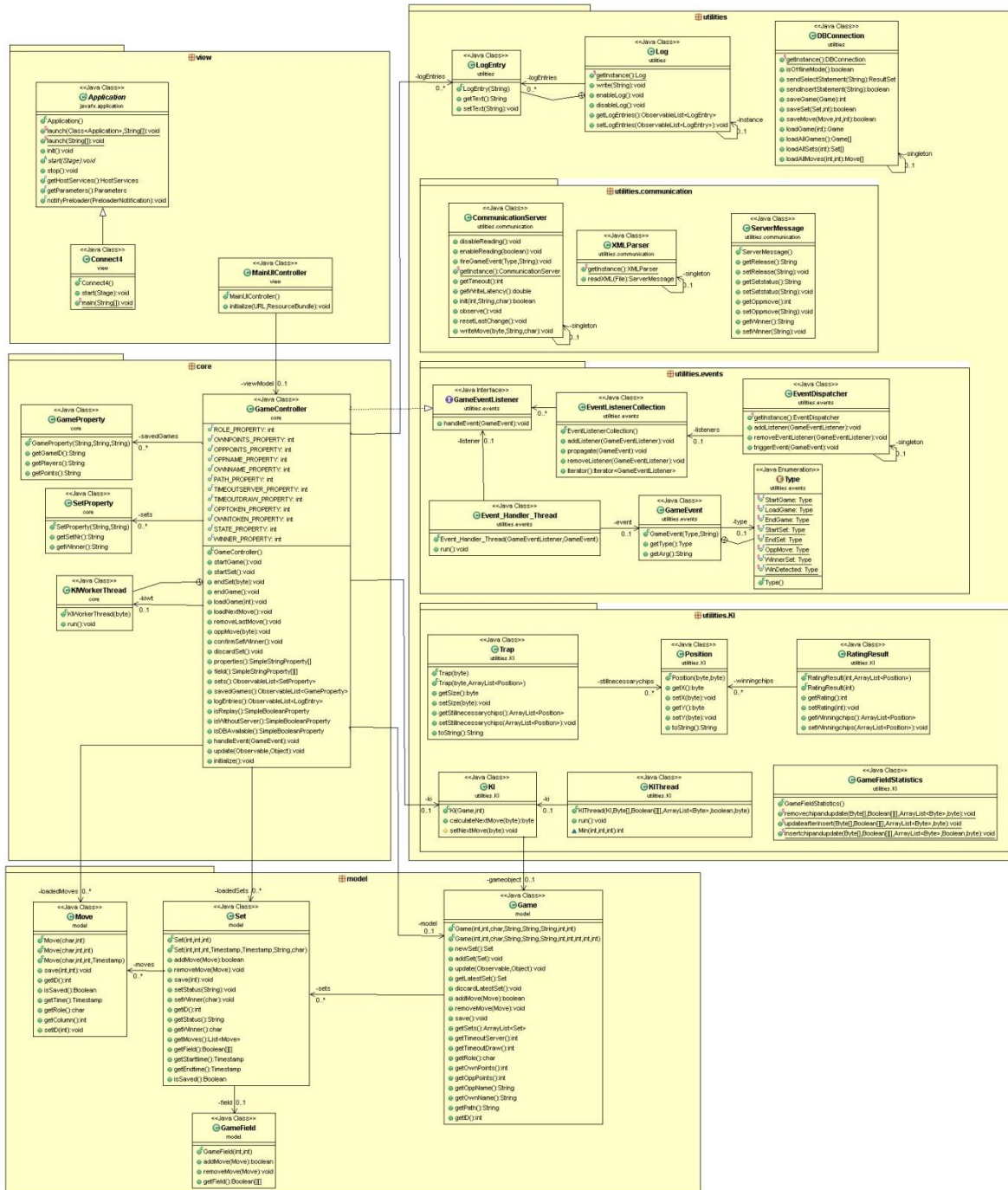
### B.3. Programmablauf

Die wichtigsten Funktionen sind als Sequenzdiagramm abgebildet im Anhang beigelegt. Dadurch werden die zugrunde liegenden Abhängigkeiten abgebildet. 5 Sequenzdiagramme sind auf der CD beigelegt:

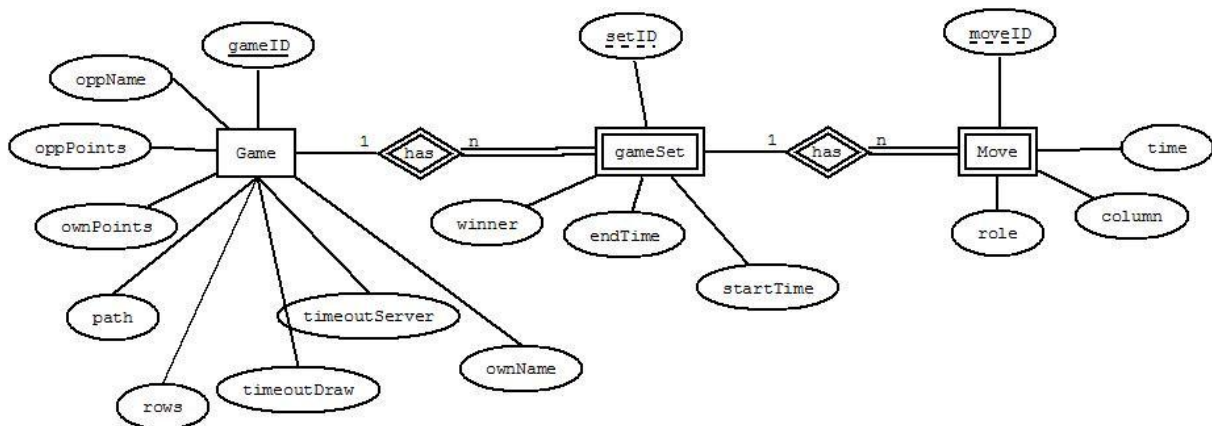
1. Programm starten: Beim Start des Programms wird in zunächst die Benutzungsoberfläche aufgebaut. Dazu werden die JavaFX-spezifischen Elemente (Stage, FXMLLoader, Scene) geladen. Ebenso werden der MainUIController und der GameController als steuernde Elemente geladen.
2. Spiel starten: Beim Starten 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.

## B.4. Anhang

### B.4.1. Klassendiagramm



### B.4.2. Entity-Relationship Diagramm



### B.4.3. Javadoc

Ein Javadoc mit der Beschreibung der Klassen und Methoden sowie weiterer wichtiger Parameter befindet sich im Anhang. Über den Aufruf der „index.html“ kann diese komfortabel eingesehen werden.

## C. Projektdurchführung

### C.1. Entwicklungsmodell und Testkonzept

#### C.1.1. 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
  - 1.1. Ziele: erfolgreichen Prototypen erstellen; dazu: gemeinsame Überlegungen zur Architektur und daraufhin Arbeit in Kleingruppen an vereinbarten Arbeitspaketen; Erstellung eines Zeitplans
  - 1.2. 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
- 1.3. 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 erfolgt gemäß den Zuständigkeiten. Testen des Prototypen erfolgt manuell.

#### 1.4. 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

## 2. Beta-Release

- 2.1. 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.
- 2.2. Vorgehen: Wie bisher hat jeder seinen Aufgabenbereich, da sich diese Aufteilung bewährt hat. Für Absprachen werden die Vorlesungsstunden genutzt.
- 2.3. Produkt: (vgl. Quellcode) Testklassen und -fälle z.T. werden bereits implementiert und entworfen.
- 2.4. 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 noch weiterentwickelt werden; die Stabilität des Programms muss ausgebaut werden.

## 3. Release:

- 3.1. Ziele: Ziel ist die Abgabe des voll funktionsfähigen Programms, sowie der Dokumentation.
- 3.2. 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.
- 3.3. Produkt: Als Produkte können der Agent, die Dokumentation, ebenso wie die Tests angesehen werden.
- 3.4. 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.

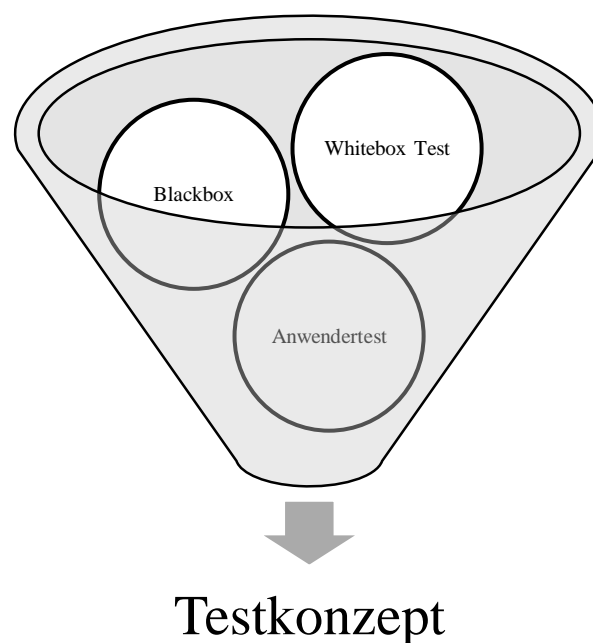
### C.1.2. Testkonzept

Um die Qualität unserer Software sicherzustellen ist das Testen der erstellten Software essentielle. Im Folgenden soll unser Konzept und die Durchführung beschrieben werden.

#### C.1.2.1. Überblick

Im Rahmen der Erstellung sind verschiedene Stufen des Testens zu durchlaufen. Diese werden nun kurz vorgestellt und einzelne Beispiele genauer erläutert.

1. Whitebox Test – Code Review
2. Blackbox Test – Testklassen
3. Anwendertest





### 1 - Whitebox Test – Code Review

Ein wichtiger Bestandteil unseres Testkonzeptes ist der Whitebox Test. Dazu haben wir jedes Codesegment durch ein weiteres Projektmitglied prüfen lassen. Dabei wurde die Funktionsweise erläutert und die Performance, Erweiterbarkeit und Notwendigkeit besprochen.

Die Ergebnisse dieser Tests wurden umgehend umgesetzt. Im Laufe der Entwicklungszeit konnten somit erhebliche Verbesserungen erzielt werden.

Die Testklassen finden Sie auf der mitgelieferten CD.

### 2 - Blackbox Tests

Eine wirkliche Aussage der Qualität der Software ist jedoch nur mit einer Vielzahl von Tests mit den unterschiedlichsten Eingabeparametern möglich. Dazu ist eine Automatisierung notwendig. Diese haben wir über Testklassen realisiert.

Folgende Komponenten wurden dabei getestet:

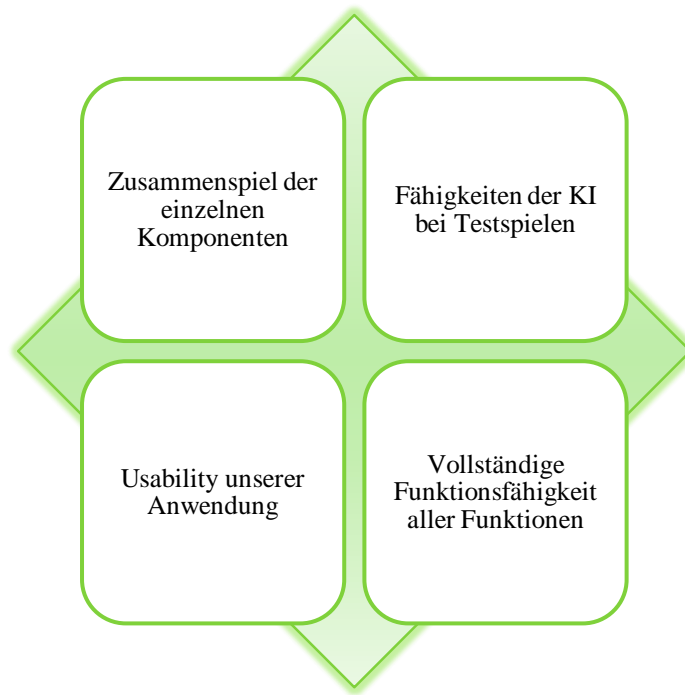
- CommunicationServer
- DBConnector
- EventHandler
- Log
- Abfrage der DB

Durch diese Testklassen wurden viele Schwachstellen aufgedeckt und konnten behoben werden. Insbesondere die Ausnahmebehandlung für die verschiedenen Problemfälle konnten maßgeblich verbessert werden.

### 3- Anwendertest

Der dritte Teil unseres Testkonzeptes umfasst die Anwendertests. Sie waren mit Abstand die wichtigsten, da sie die Integrationstests unserer Software darstellen. Durch sie konnten folgende Punkte überprüft werden:





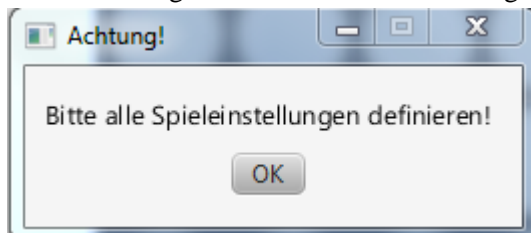
Dabei haben wir folgende Testcases definiert, die von unabhängigen Usern durchgeführt wurden und deren Feedback wir verarbeitet haben.

### *Case 1 – Eingabefelder prüfen*

1. Programm starten
2. Rolle O setzen
3. Eigenen Namen eingeben
4. Gegnername leer lassen
5. Verzeichnispfad C:\test auswählen
6. Timeout Zeiten belassen



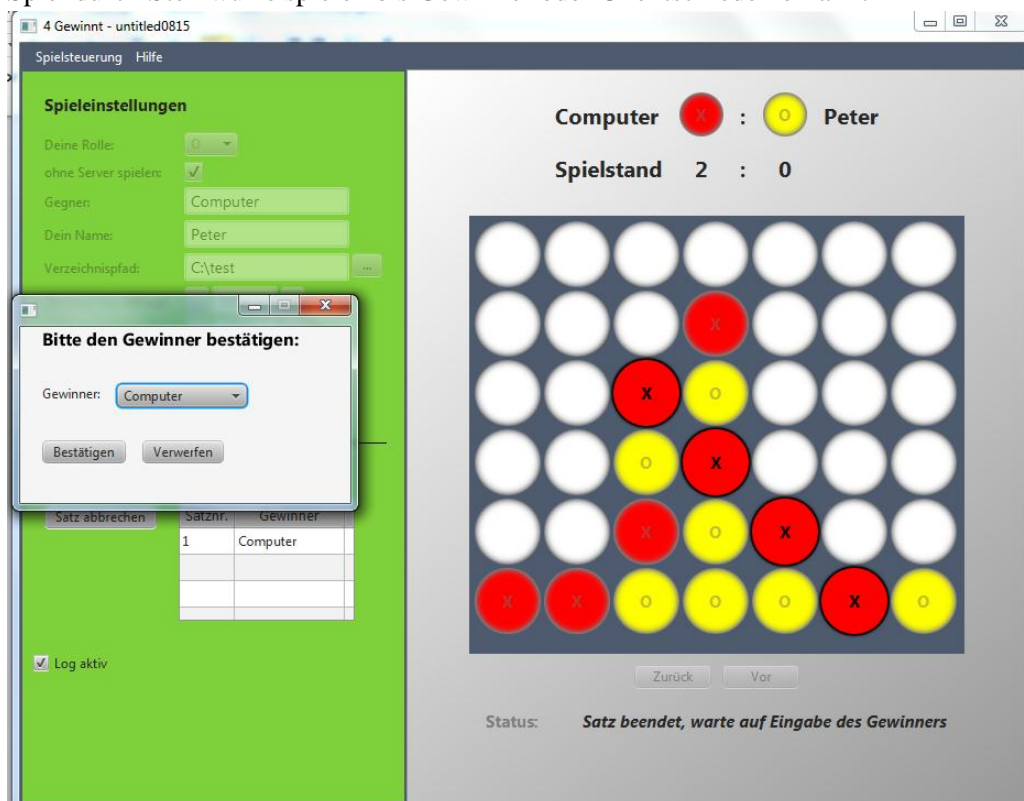
7. Spiel starten drücken
8. Fehlermeldung erscheint → Fehlermeldung bestätigen



9. Gegnername ausfüllen
10. Spiel starten drücken
11. Spiel abbrechen drücken
12. Programm über Reiter Spielsteuerung beenden

### Case 2 – Lokal Spielen

1. Programm starten
2. Felder ausfüllen (ohne Server spielen auswählen)
3. Spiel starten
4. Satz starten
5. Spiel durch Steinwürfe spielen bis Gewinner oder Unentschieden erkannt

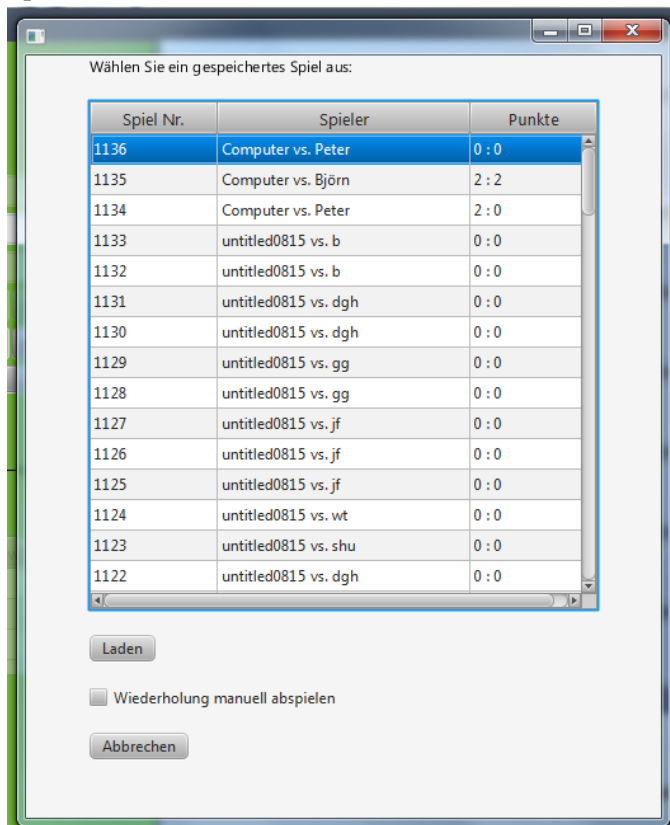


6. Sich selbst als Gewinner setzen
7. Neuen Satz starten
8. Spiel durch Steinwürfe spielen bis Gewinner oder Unentschieden erkannt
9. Computer als Gewinner setzen
10. Neuen Satz starten
11. Spiel durch Steinwürfe spielen bis Gewinner oder Unentschieden erkannt

12. Unentschieden setzen
13. Spiel beenden

### Case 3 – Spiel laden

1. Programm starten
2. Spiel laden drücken



3. Letztes Spiel wählen
4. Laden drücken
5. Satz spielen drücken
6. Satz abbrechen drücken
7. Verwerfen drücken
8. Spiel beenden drücken
9. Programm schließen

## C.2. Javadoc

### 3.4.1. 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

### 3.4.2. Javadoc Anleitung

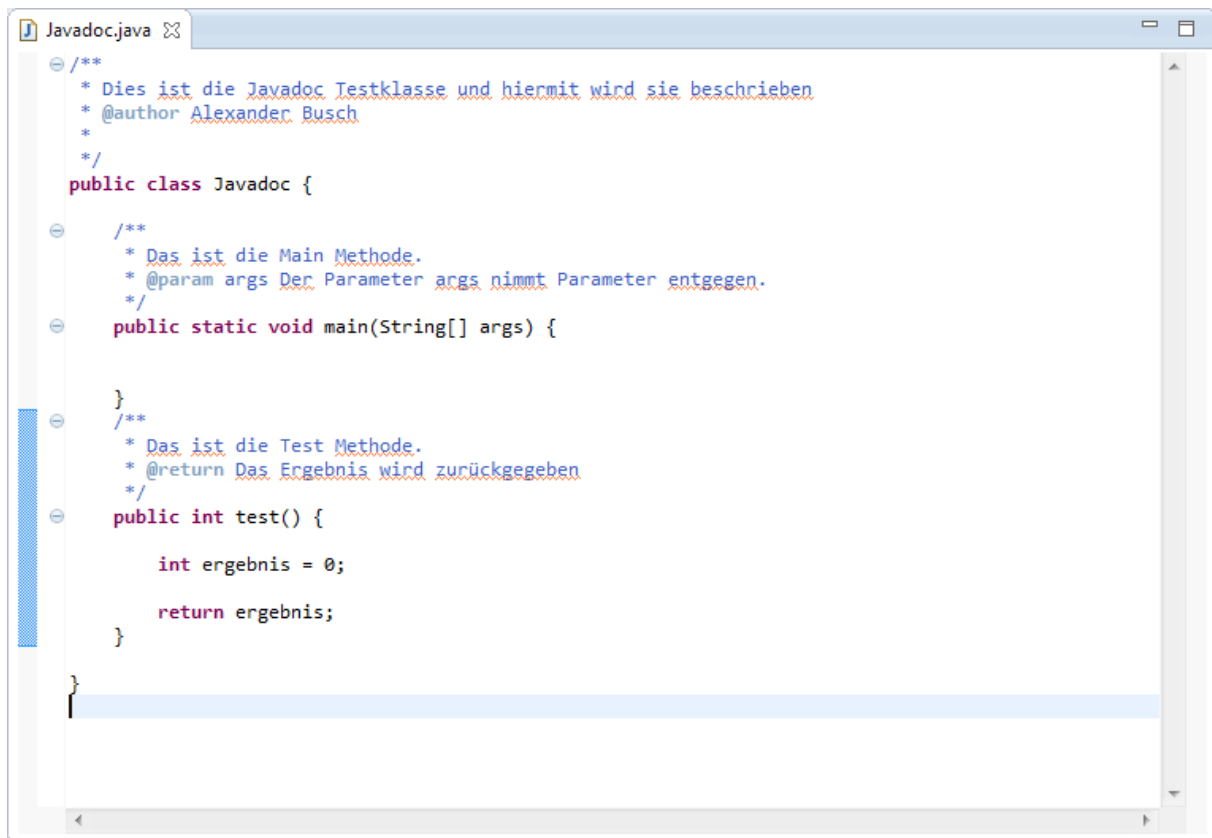
**Voraussetzung:** Um Javadoc nutzen zu können muss diese Anwendung auf dem Rechner verfügbar sein. Dies bedeutet, dass sich im entsprechenden Java-Verzeichnis eine javadoc.exe befinden muss. Der Pfad lautet in diesem Fall [Pfad zum Java-Verzeichnis]\bin\.

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.



```
1  /**
2   * Dies ist die Javadoc Testklasse und hiermit wird sie beschrieben
3   * @author Alexander Busch
4   */
5  public class Javadoc {
6
7      /**
8       * Das ist die Main Methode.
9       * @param args Der Parameter args nimmt Parameter entgegen.
10      */
11      public static void main(String[] args) {
12
13      }
14
15      /**
16       * Das ist die Test Methode.
17       * @return Das Ergebnis wird zurückgegeben
18      */
19      public int test() {
20
21          int ergebnis = 0;
22
23          return ergebnis;
24      }
25  }
```

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 Ausgabeverzeichnis, 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.

---

## Class Javadoc

java.lang.Object  
Javadoc

---

```
public class Javadoc
extends java.lang.Object
```

Dies ist die Javadoc Testklasse und hiermit wird sie beschrieben

### Author:

Alexander Busch

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

## Method Summary

### Methods

Modifier and Type	Method and Description
static void	<code>main(java.lang.String[] args)</code> Das ist die Main Methode.
int	<code>test()</code> Das ist die Test Methode.

### Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Die verwendeten Methoden werden samt Erläuterung angezeigt.

## Method Detail

### main

```
public static void main(java.lang.String[] args)
```

Das ist die Main Methode.

#### Parameters:

args - Der Parameter args nimmt Parameter entgegen.

### test

```
public int test()
```

Das ist die Test Methode.

#### Returns:

Das Ergebnis wird zurückgegeben

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

## C.3. Git

Bei Git<sup>2</sup> 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<sup>3</sup> ausgewählt. Github bietet seine Dienste kostenfrei an. Die Tatsache, dass Projekte wie PHP, Ruby oder JUnit auf GitHub weiterentwickelt werden förderte die Entscheidung diesen Anbieter auszuwählen.

---

<sup>2</sup> <http://git-scm.com/>

<sup>3</sup> <https://github.com/>

Um den direkten Datenaustausch aus Eclipse heraus zu ermöglichen wurde das Plugin EGit<sup>4</sup> verwendet. EGit ist das am häufigsten genutzte Plugin zur Nutzung von Git in Eclipse und daher der de-facto Standard.

## C.4. Databinding

Eine Möglichkeit, Verknüpfungen von Daten zu schaffen, ist das Binden von Daten an Properties, auch Data Binding genannt. Properties sind Objekte die das Property Interface implementieren welches unter anderem aus dem Observable Interface besteht. Die Idee ist, dass sich andere Objekte, die selber das Property oder Observer Interface implementieren, an Properties registrieren und automatisch bei Änderungen informiert werden und entweder den Wert übernehmen oder mittels Converter umwandeln. Verwendet wird diese Technik um das UI mit Daten zu versorgen. Anstatt dies über Methoden im Codebehind zu realisieren, werden die Properties im UI mit den Properties vom Datenmodell oder Controller verbunden (Mehr dazu unter Entwurfsmuster). Beispielsweise wird die TextProperty eines Textfeldes an das Feld ownPoints gebunden:

```
ownPoints.textProperty().bind(viewModel.ownPoints());
```

Außerdem wird der im Root-Element der FXML-Struktur angegebene Controller über die FXML-Struktur vererbt. Somit können auch alle darunterliegenden Elemente darauf zugreifen und müssen bei Verwendung nur noch den Pfad zu dem benötigten Feld oder der Methode angeben.

---

<sup>4</sup> <http://www.eclipse.org/egit/>