# Änderungshistorie

**Änderungshistorie**

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Status | Änderungen |
| 28.09.2015 | 0.1 | draft | Kapitel *Projektziel* verfasst |
| 05.10.2015 | 0.2 | draft | Kapitel 2 und 3 verfasst, optische Anpassungen |
| 12.10.2015 | 0.3 | draft | Kapitel 4 verfasst |
| 01.11.2015 | 0.4 | draft | Kapitel 5 und 6 verfasst |
| 02.11.2015 | 0.5 | draft | Kleine Anpassungen am Dokument |
| 06.11.2015 | 0.6 | draftl | Finalisierung des Dokumentes |
| 07.11.2015 | 1.0 | final | Finaler Review |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Inhaltsverzeichnis

**Inhaltsverzeichnis**

[Änderungshistorie i](#_Toc434239626)

[Inhaltsverzeichnis I](#_Toc434239627)

[Abbildungsverzeichnis III](#_Toc434239628)

[Tabellenverzeichnis IV](#_Toc434239629)

[Anlagenverzeichnis V](#_Toc434239630)

[1. Einleitung 1](#_Toc434239631)

[1.1 Projektziel 1](#_Toc434239632)

[1.2 Vorgehensmodell Softwareentwicklung 2](#_Toc434239633)

[2. Anforderungen 4](#_Toc434239634)

[3. Analyse 6](#_Toc434239635)

[3.1 Systemkontext 6](#_Toc434239636)

[3.2 Ermittlung der Projekt-Anforderungen 7](#_Toc434239637)

[3.2.1 Geschäftsanwendungsfälle 7](#_Toc434239638)

[3.2.2 Systemanwendungsfälle 9](#_Toc434239639)

[3.3 Spezifikation der Projekt-Anforderungen 10](#_Toc434239640)

[3.3.1 Funktionale Anforderungen 10](#_Toc434239641)

[3.3.2 Nichtfunktionale Anforderungen 10](#_Toc434239642)

[4. Entwurf 12](#_Toc434239643)

[4.1 System-Architektur 12](#_Toc434239644)

[4.2 Verhalten des Systems 13](#_Toc434239645)

[4.2.1 Aktivitätsdiagramme 13](#_Toc434239646)

[4.2.2 Zustandsdiagramme 13](#_Toc434239647)

[4.3 Logische Komponenten des Systems 14](#_Toc434239648)

[4.3.1 Datenbank 14](#_Toc434239649)

[4.3.2 Server-Schnittstellen 14](#_Toc434239650)

[4.3.3 File-Schnittstelle 14](#_Toc434239651)

[4.3.4 Pusher-Schnittstelle 15](#_Toc434239652)

[4.3.5 Grafische Benutzeroberfläche 15](#_Toc434239653)

[4.3.6 Künstliche Intelligenz 20](#_Toc434239654)

**Inhaltsverzeichnis**

[5. Implementierung 22](#_Toc434239655)

[5.1 Java-API Dokumentation 22](#_Toc434239656)

[5.2 (Sequenzdiagramm) 22](#_Toc434239657)

[6. Test 23](#_Toc434239658)

[6.1 Testverfahren 23](#_Toc434239659)

[6.2 Testfälle 24](#_Toc434239660)

[7. Inbetriebnahme 27](#_Toc434239661)

[Anhang 28](#_Toc434239662)

# Abbildungsverzeichnis

**Abbildungsverzeichnis**

[**Abbildung 1: Vorgehensmodell „Wasserfallmodell“** 2](#_Toc434241901)

[**Abbildung 2: Systemkontext** 6](#_Toc434241902)

[**Abbildung 3: System-Architektur** 12](#_Toc434241903)

[**Abbildung 4: ER-Diagramm der Datenbank** 14](#_Toc434241904)

[**Abbildung 5: Durchgeführte Testverfahren** 23](#_Toc434241905)

# Tabellenverzeichnis

**Tabellenverzeichnis**

[**Tabelle 1: Geschäftsanwendungsfall 1** 7](#_Toc434241911)

[**Tabelle 2: Geschäftsanwendungsfall 2** 7](#_Toc434241912)

[**Tabelle 3: Geschäftsanwendungsfall 3** 8](#_Toc434241913)

[**Tabelle 4: Geschäftsanwendungsfall 4** 8](#_Toc434241914)

[**Tabelle 5: Geschäftsanwendungsfall 5** 9](#_Toc434241915)

[**Tabelle 6: Systemanwendungsfall 1.2** 9](#_Toc434241916)

[**Tabelle 7: Funktionale Anforderungen** 10](#_Toc434241917)

[**Tabelle 8: Nichtfunktionale Anforderungen** 11](#_Toc434241918)

[**Tabelle 9: Testfall 1 Teilnahme am Turnier** 24](#_Toc434241919)

[**Tabelle 10: Testfall 2 Sieger des Satzes manuell bestimmen** 25](#_Toc434241920)

[**Tabelle 11: Testfall 3 Sieger des Spieles manuell bestimmen** 25](#_Toc434241921)

[**Tabelle 12: Testfall 4 Historie anzeigen** 26](#_Toc434241922)

# Anlagenverzeichnis

**Anlagenverzeichnis**

[Anlage 1: Projektplan 28](#_Toc434241923)

# Einleitung

**Einleitung**

## Projektziel

Dieses Dokument wurde im Rahmen der Veranstaltung „WI-Projekt“ der DHBW Mannheim, gehalten von Prof. Dr. Lauterbach (Im Folgenden als *Auftraggeber* bezeichnet), von den Studierenden Cara Damm, Felix Hedrich, Majken Plügge, Alexander Kern und Björn Korioth (im Folgenden als *Projektteilnehmer* bezeichnet) verfasst. Es handelt sich dabei um das Ergebnis aller mit der Veranstaltung in Beziehung stehender schriftlichen Ausarbeitungen. Die Ausgangssituation stellt die übergeordnete Aufgabe dar, mit Hilfe des in den vorherigen Semestern erlernten Fachwissens einen autonom spielfähigen Software-Agenten für das weltweit bekannte Spiel „Vier-Gewinnt“ zu entwickeln. Die Vorgaben sind dabei auf wesentliche Eigenschaften beschränkt – sowohl das Projektmanagement als auch die Auswahl geeigneter Tools und einer Methodik liegen in der Verantwortung der Projektteilnehmer.

Das Projektziel ist durch die Anforderungen des Auftraggebers als fix anzusehen und wird im Wesentlichen durch drei vorgegebene Abgaben definiert, die zu konkreten Vorlesungsterminen bereitgestellt werden müssen. Dabei handelt es sich um:

* **Agent** Prototyp (4. Vorlesungstermin)
* **Agent** Beta (6. Vorlesungstermin)
* **Agent** Release (letzter Vorlesungstermin)
* Die hier vorliegende **Dokumentation** (letzter Vorlesungstermin)
* Erfolgreiche **Turnierteilnahme** des Software-Agenten (letzter Vorlesungstermin)

Ziel dieses Dokumentes ist es, dem Leser einen umfangreichen Überblick über die Entwicklungsphasen der Software einerseits und über die technischen Komponenten andererseits zu vermitteln. Dadurch soll gewährleistet werden, dass eine unproblematische (Weiter-)Entwicklung der Software auch zukünftig möglich ist und für dritte Parteien, insbesondere den Auftraggeber, die bisherigen Entwicklungs- und Design-Entscheidungen, die von den Projektteilnehmern getroffen wurden, zu jeder Zeit nachvollziehbar sind.

Es sei darauf hingewiesen, dass im Lieferumfang neben dieser Dokumentation auch das Benutzerhandbuch.pdf sowie die Projektplanung.pdf beiliegen.

## Vorgehensmodell Softwareentwicklung

**Einleitung**

Die Wahl eines geeigneten Vorgehensmodells wurde den Projektteilnehmern überlassen. Daher erschien es zu Projektbeginn als äußerst sinnvoll, ein solches Vorgehensmodell zu definieren, das allen Projektteilnehmern zusagt und mit den Rahmenbedingungen (Anzahl Teilnehmer, Anzahl geschätzter Unterrichtseinheiten zur Durchführung des Projektes, Komplexität) harmonisiert. In einer engeren Auswahl standen die folgenden drei, auch in der einschlägigen Literatur und Best Practices vieler Software-Projekte vorgeschlagenen, Varianten Wasserfallmodell, Scrum und Extreme Programming.

Die finale Entscheidung fiel nach gemeinsamer Diskussion dann auf das Vorgehensmodell „Wasserfallmodell“, da es sich um ein bewährtes Modell in der Software-Entwicklung handelt, alle Projektteilnehmer damit bereits die meiste Erfahrung sammeln konnten und es Sicht der Projektteilnehmer auch gut mit den vorgegebenen Rahmenbedingungen harmoniert. Abbildung 1 zeigt den chronologischen Ablauf dieses Modells auf:



**Abbildung 1: Vorgehensmodell „Wasserfallmodell“**

Die Gliederung dieser technischen Dokumentation wurde in starker Anlehnung an die in Abbildung 1 aufgezeigten Phasen des Modells angelehnt:

* Alle **Anforderungen**, die vom Auftraggeber als fix vorgegeben wurden, werden in einem ersten Kapitel beschrieben.
* In der **Analysephase** soll, abgeleitet aus den Anforderungen, der Systemkontext der Anwendung dargestellt werden und zusätzlich sämtliche relevante Geschäfts- und Systemanwendungsfälle gebildet werden.
* In der **Entwurfsphase** sollen die wesentlichen Design-Entscheidungen aufgezeigt werden. Die Software wird auf unterschiedlichen Ebenen modelliert und grafisch dargestellt. Hier finden sich ebenfalls Anmerkungen zur Funktionsweise der wichtigsten Komponenten (beispielsweise zur GUI, KI und Datenbank), die von hoher Bedeutung für die Funktionalität der Anwendung sind.

**Einleitung**

* In der hauptsächlich praktisch durchgeführten **Implementierungsphase** werden die Entwurfsentscheidungen in Quellcode umgesetzt. Die technische Dokumentation stellt hier Informationen zum Quellcode in Form einer Java-API zur Verfügung.
* Ein abschließendes Kapitel wird die verwendete Methodik der **Testphase** erläutern, welche vor der Turnierteilnahme durchlaufen wurde.
* Als **Inbetriebnahme** wird lediglich die Teilnahme am Turnier in der abschließenden Vorlesung gewertet.
* Auf eine spätere **Wartung** der Anwendung wird verzichtet, da das Projekt zeitlich auf ein Semester limitiert ist. Diese wird daher nicht Teil dieser technischen Dokumentation sein.

Das gewählte Vorgehensmodell spiegelt sich ebenfalls in der Projektplanung wider. Hier sei auf das beiliegende Dokument Projektplanung.pdf verwiesen. Die Projektplanung wurde aus Gründen der Übersichtlichkeit aus der rein technischen Dokumentation ausgegliedert und kann dort eingesehen werden.

# Anforderungen

**Anforderungen**

In diesem Kapitel sollen die Minimalanforderungen des Auftraggebers an das Produkt einmal lastenheftartig erfasst werden. Auf diesen Anforderungen basiert die Entwicklungsarbeit der nachfolgenden Kapitel dieser technischen Dokumentation.

**Allgemein**

Im Rahmen des Entwicklungsauftrages soll eine Software entwickelt werden, die es ermöglicht, das Spiel „Vier gewinnt“ auszutragen. Die Besonderheit liegt in der Entwicklung eines Agenten, der eine Partie dieses Spiels autonom ohne Einwirkung des Benutzers mit einem möglichst zufriedenstellenden Ergebnis austragen kann. Der Agent muss dazu mit einem bereitgestellten Server-Programm kommunizieren, mit dem auch ein späterer Gegenspieler, d.h. der Agent eines anderen Teams, verbunden ist.

**Benutzerschnittstelle**

Das Programm muss dazu über eine grafische Benutzeroberfläche (Benutzerschnittstelle) verfügen, die mindestens den Satzstatus, den Spielstand, das Spielfeld sowie alle Züge des aktuellen Spieles auf dem Spielfeld anzeigt. Zur Umsetzung muss als verwendete Technologie zur Umsetzung Swing oder alternativ JavaFx herangezogen werden.

**Serverschnittstelle**

Zur Kommunikation zwischen Agent und Server müssen zwei Schnittstellen implementiert werden. Dabei handelt es sich um die Datei-Schnittstelle und die Push-Schnittstelle. Die Kommunikation erfolgt immer nur über jeweils eine Schnittstelle. Während eines Satzes erfolgt kein Wechsel der Schnittstelle – zwischen den Sätzen, d.h. also auch während eines Spieles, muss die Schnittstelle jedoch gewechselt werden können. Technologisch erfolgt die Kommunikation von Server und Agents

* bei der Datei-Schnittstelle über Dateien auf Basis von Streams.
* bei der Push-Schnittstelle über Events auf der Basis von WebSockets.

**Datenbank**

**Anforderungen**

Es muss eine Datenbank für die Datenhaltung implementiert werden. Es muss eine automatische Speicherung der benötigten Daten durch den Agenten erfolgen. Bei diesen zu speichernden Daten handelt es sich mindestens um Gegner, Startspieler, Sieger, Punkte, Spiele, Sätze und Züge. Die technologische Realisierung erfolgt auf der Basis von HSQLDB. Weiterhin besteht eine Minimalanforderung zur Abfrage der Datenbankeinträge: So müssen alle gespielten Spiele und alle Züge eines frei wählbaren Spiels der angezeigten Spiele angezeigt werden können.

Für weitere Details wird auf das online abrufbare Dokument Anforderungsbeschreibung[[1]](#footnote-1) aus der Vorlesung WI-Projekt verwiesen.

# Analyse

**Analyse**

## Systemkontext

Der Systemkontext der Anwendung „Vier gewinnt“ wurde aus den Anforderungen abgeleitet. Wie in Abbildung **X** aufgezeigt, hat das System eine Schnittstelle zum User (etwa grafische Anzeige des Spielfeldes, Zugriff auf die Anwendung durch Eingabegeräte wie Tastatur und Maus), zur Datenbank (lokale Speicherung der relevanten Datensätze) und die Schnittstelle zum Server (entweder über die vorher auswählbare File-Schnittstelle oder über die Verwendung der Pusher-Schnittstelle.)



**Abbildung 2: Systemkontext**

## Ermittlung der Projekt-Anforderungen

## Geschäftsanwendungsfälle

In diesem Kapitel befinden sich sämtliche erarbeitete Geschäftsanwendungsfälle, die beim Ausführen des Programmes durchlaufen werden können sollen. Es handelt sich dabei um X,X,X,X,X,X und X.

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 1.1 |
| Art | Geschäftsanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 1: Geschäftsanwendungsfall 1**

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 2 |
| Art | Geschäftsanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 2: Geschäftsanwendungsfall 2**

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 3 |
| Art | Geschäftsanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 3: Geschäftsanwendungsfall 3**

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 4 |
| Art | Geschäftsanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 4: Geschäftsanwendungsfall 4**

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 5 |
| Art | Geschäftsanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 5: Geschäftsanwendungsfall 5**

## Systemanwendungsfälle

|  |  |
| --- | --- |
|  | |
| Name |  |
| Nummer | 1.2 |
| Art | Systemanwendungsfall |
| Kurzbeschreibung |  |
| Auslöser |  |
| Ergebnis |  |
| Akteure |  |
| Eingehende Informationen |  |
| Vorbedingungen |  |
| Nachbedingungen |  |
| Ablauf |  |

**Tabelle 6: Systemanwendungsfall 1.2**

## Spezifikation der Projekt-Anforderungen

## Funktionale Anforderungen

Funktionale Anforderungen beschreiben per Definition, was ein zu entwickelndes System später bei Inbetriebnahme leisten können muss beziehungsweise welche Funktionen es aufweisen muss, damit ein Anwender ein vorgegebenes Problem damit lösen kann.[[2]](#footnote-2)

Ein erheblicher Teil der nun aufgeführten Anforderungen ist auf Vorgaben des Auftraggebers zurückzuführen. Einige Anforderungen liegen jedoch auch in logischen Schlussfolgerungen benötigter Funktionalitäten aus den in Kapitel 3.1.1 und 3.1.2 ermittelten Use Cases begründet, ohne die das Programm nicht oder kaum benutzerfreundlich zu verwenden ist. Dies betrifft insbesondere Aspekte der grafischen Benutzeroberfläche. Aus diesem Grunde wurde eine Klassifikation vorgenommen, die die Herkunft der Anforderungen eingrenzen soll. Erstere, vom Auftraggeber bestimmte Anforderungen, werden als „muss“, letztere, selbst für notwendig erachtete Anforderungen, als „kann“ klassifiziert.

Die folgenden in Tabelle **X** gelisteten Punkte bilden die Gesamtheit aller funktionaler Anforderungen und bestimmen im Wesentlichen den Funktionsumfang des Tools.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Bezeichnung | Beschreibung | Klassifikation |
| 1 |  |  |  |
| 1.1 |  |  |  |
| 1.2 |  |  |  |
| 1.3 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

**Tabelle 7: Funktionale Anforderungen**

## Nichtfunktionale Anforderungen

Nach Helmut Balzert legt eine nichtfunktionale Anforderung „(…) eine vom Softwaresystem oder einer seiner Komponenten bereitzustellende Funktion oder bereitzustellenden Service fest“.[[3]](#footnote-3) Auch für das Programm *Vier gewinnt* ergibt sich eine Reihe von kritischen nichtfunktionalen Anforderungen, die nun im Folgenden mit der Tabelle **X** aufgeführt werden. Um diese gerecht kategorisieren zu können, wurde die Entscheidung getroffen, die Norm ISO/IEC 9126 zu referenzieren, aus der sich folgende sechs Kategorien ergeben und die als Kategorisierung der nichtfunktionalen Anforderungen verwendet werden:

**Funktionalität**, **Zuverlässigkeit**, **Benutzbarkeit**, **Effizienz**, **Wartbarkeit**, **Portabilität**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Bezeichnung | Beschreibung | Klassifikation |
| Funktionalität | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 | Funktionalität |  |  |
| Zuverlässigkeit | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Benutzbarkeit | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Effizienz | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Wartbarkeit | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| Portabilität | | | |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

**Tabelle 8: Nichtfunktionale Anforderungen**

# Entwurf

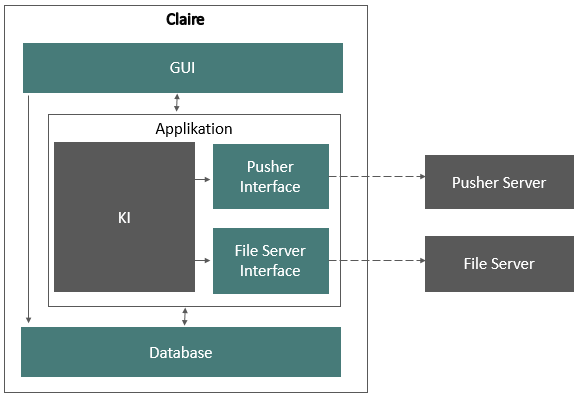
**Entwurf**

## System-Architektur

Der „Claire’s Vier gewinnt“-Agent Claire ist nach einer klassischen 3-Schichten-Architektur aufgebaut. Somit besteht die Architektur aus den folgenden drei Bausteinen:

* Graphische Oberfläche
* Hauptapplikation
* Datenbank

Die untenstehende Abbildung **X** zeigt die System-Architektur schematisch auf.



**Abbildung 3: System-Architektur**

Die graphische Oberfläche (GUI) bekommt die anzuzeigenden Informationen von der Applikationsebene weitergeleitet. Das gleiche gilt auch für die Datenbank (Database).

Die Applikationsebene selber besteht aus 3 Teilen. Zum einen die künstliche Intelligenz (KI), die die zu setzende Zeile berechnet, zum anderen aber auch die 2 Kommunikationsschnittstellen. Diese sind sowohl eine Schnittstelle zur Kommunikation mit einem Pusher-Server, als auch eine Schnittstelle zur Kommunikation über ein Fileablage-System.

## Verhalten des Systems

**Entwurf**

## Aktivitätsdiagramme

## Zustandsdiagramme

## Logische Komponenten des Systems

**Entwurf**

## Datenbank

Die folgende Abbildung **X** zeigt anschaulich das Datenbankmodell der Anwendung.



**Abbildung 4: Diagramm der Datenbank**

Die Datenbank wurde so modelliert, dass eine gegnerische Person n Spiele gegen die künstliche Intelligenz Claire spielen kann, ein Spiel besitzt n Sätze und ein Satz besitz n Züge. In dem Paket Datenbank befindet sich eine Klasse für die Datenbank, welche den Zugriff auf die Datenbank mittels der SQL-Befehle abbildet. Des Weiteren kümmern sich drei zusätzliche Klassen darum, dass auf die Daten des aktuellen Spielers, des aktuellen Spiels und des aktuellen Satz zugegriffen werden kann. Dies geschieht mittels Speicherung der Werte in die statischen Klassenvariablen.

## Server-Schnittstellen

## File-Schnittstelle

* Auswahl eines Ordners, der zur Kommunikation zwischen den beiden Clients und dem Server dient
* Auf Client wird Team (O oder X) ausgewählt
* Anhand des Team-Namens werden die Dateien zur Kommunikation der zwei Teams erstellt
* Server kann anhand dieser Dateien (deren Namen) feststellen, welches Team welchen Zug durchführt

**Entwurf**

* Workflow:
* Server generiert XML-Datei (siehe Glossar Herr. Lauterbach)
* Client liest XML-Datei ein und setzen evtl. Gegnerzug in unsere GUI
* Client-KI erzeugt einen Zug und Anzeige in GUI
* Server-Schnittstelle erstellt Datei mit dem durchzuführenden Zug (Koordinate) und legt diese auf dem vorher in den Einstellungen definierten Ordnerpfad ab

## Pusher-Schnittstelle

* Es gibt einen Pusher-Server, Verbindung Client – Server, sämtliche Kommunikation erfolgt über Pusher
* Auf Pusher-Server existiert App (besondere Credentials – Adresse (Account)
* Innerhalb des Pushers erfolgt die Kommunikation über Channels, für diese Channel-Kommunikation wird eine Verschlüsselung verwendet, ein secretkey und gehashed

 Innerhalb der Channels existieren events, auf die reagiert werden kann

* MoveToAgent: Sämtliche vom Server ausgehenden Befehle / Kommunikation
* Client empfängt MoveToAgent-Nachricht als Json-Format (Client kann Informationen verarbeiten)
* Workflow wie bei File-Schnittstelle
* Client-Event: Sämtliche vom Client ausgehenden Befehle (Zug setzen, …)

## Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche basiert auf der JavaFX Technologie. Hierbei handelt es sich um eine Kombination aus FXML Seiten und Java. Durch die Verwendung von FXML Seiten wurde das Model-View-Controller Konzept realisiert.

Die Oberfläche wird in den FXML Seiten beschrieben. Somit kann mit einer XML basierten Sprache die Architektur der anzuzeigenden Seite aufgebaut werden.

Controller ermöglichen, dass auf die Seite Einfluss genommen werden kann. So lassen sich während der Programmlaufzeit Veränderungen durchführen.

Außerdem kontrollieren die Controller die Aktionen, die von der Benutzeroberfläche aus angestoßen werden.

In diesem Kapitel werden die folgenden Unterpunkte der grafischen Benutzeroberfläche ausführlicher beschrieben:

**Entwurf**

* Aufbau einer FXML-Seite
* Eine FXML-Seite anzeigen
* Aktionen auf der Benutzeroberfläche ausführen
* Oberflächenanpassungen während der Programmlaufzeit
* Tabellenansichten

**Aufbau einer FXML Seite:**

Der Aufbau einer FXML Seite ist festgelegt. Zunächst wird ein Container definiert, in dem anschließend Controls, Menues, Chapes und Chats hinzugefügt werden können.

In Zeile 8 wird der Controller definiert, zunächst wird hier angegeben welcher Container verwendet wurde, wie in diesem Fall: „Pane“. Außerdem wird in der Controller Definition festgelegt wie groß und wie breit das anzuzeigende Fenster sein soll (Zeile 9). Ebenfalls ermöglicht uns das Verwenden von FXML-Seiten, dass wir CSS-Dateien verwenden können. Welches Stylesheet verwendet werden soll, muss dem Container mitgeteilt werden, dies wird in Zeile 10 gemacht.

Außerdem muss in FXML-Seite definiert werden von welchem Controller diese kontrolliert wird (Zeile 13). In der Controller-Klasse kann dann anschließend Einfluss auf die FXML-Seite genommen werden.



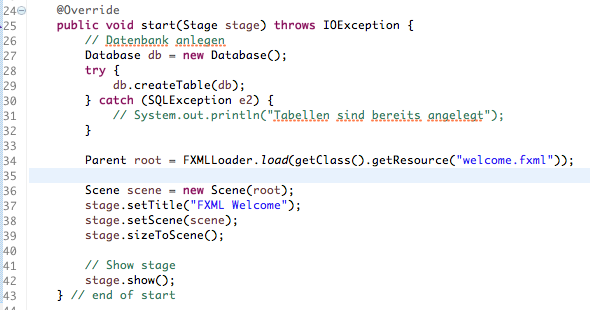
**Entwurf**

Nach der Definition des Containers werden die verschiedenen Controls, Charts, Shapes, etc. angelegt.

Zu den Controls gehört unter anderem: der Button, das Label und das ImageView. Um anschließend auf diese Felder zuzugreifen, muss Ihnen eine eindeutige ID gegeben werden. Diese ID wird mit dem Befehl *fx:id* gesetzt (Zeile 21, 22, 23, ...).

**Eine FXML Seite anzeigen**

Um eine FXML-Seite anzuzeigen, muss die gewünschte Seite zunächst geladen werden, was in Zeile 34 veranlasst wird. Wichtig zu beachten hierbei ist, bevor der FXMLLoader verwendet werden kann, muss dieser importiert werden.



Nachdem die Seite erfolgreich geladen wurde, wird eine Szene erzeugt (Zeile 36). Um eine Szene anzuzeigen, benötigt diese eine Bühne, welche anschließend die Szene übergeben bekommt (Zeile 38).

**Entwurf**

**Aktionen auf der Benutzeroberfläche ausführen**

Damit Aktionen auf der Benutzeroberfläche ausgeführt werden können, benötigt eine FXML-Seite einen Controller, der in der Containerdefinition festgelegt wird. In diesem Container müssen anschließend Importe der verwendeten Controls erzeugt werden, wie beispielsweise der Button.

**import** javafx.scene.control.Button;

Anschließend muss die Klasse „Initialize“ implementieren.

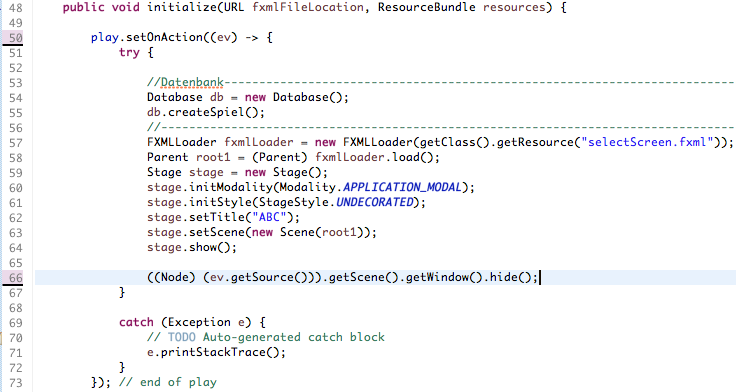
**public** **class** ControllerSpiel **implements** Initializable {

Durch dieses Implement entsteht in der Klasse die Override Methode „initialize“. Diese Methode wird ausgeführt, sobald die FXML-Seite angezeigt werden.

Hier können nun auch den verschiedenen Elementen der Benutzeroberfläche Aktionen zugeordnet werden.

Möchte man einem Button eine Funktion zuordnen, muss man diesen über seine ID ansprechen. Diese ID stammt aus der FXML-Seite. In Zeile 50 wird definiert, dass der Button eine Aktivität ausführen soll, wenn dieser gedrückt wird. Welche Aktivität ausgeführt werden soll, wird anschließend bestimmt.

In diesem Fall, soll ein neues Fenster aufgerufen werden, in dem die Einstellungen einzutragen sind. Nachdem das neue Fenster erzeugt wurde, soll das alte Fenster geschlossen werden. Dies funktioniert mit dem Aufruf in Zeile 66.



**Entwurf**

**Oberflächenanpassungen während der Programmlaufzeit**

Beim Spiel „Claire’s Vier gewinnt“ ist es notwendig, dass sich die Benutzeroberfläche während des Programmablaufs anpasst, damit die Spieler ihre Züge auf dem Spielfeld mitverfolgen können. Um dies zu ermöglichen, wird für jeden Zug ein neuer Spielstein erzeugt, der anschließend auf dem Spielfeld angezeigt wird.

ImageView iv = **new** ImageView(imageGreen);

Um das Spielfeld anzuzeigen wird ein GridPane als Controller verwendet, welches die ID „gird“ hat. Durch das Verwenden des Gridpanes ist es möglich, den neuerzeugten Spielstein an der Stelle im Feld einzutragen, welche von der Kommunikationsschnittstelle übergeben wurde.

Wie diese Kommunikation stattfindet, wird in Kapitel **X** beschrieben.

Um die Spielsteine dynamisch nach jedem Zug anzuzeigen, werden Threads verwendet, welche in Kapitel **X** verwendet werden.

grid.*setRowIndex*(iv, row);

grid.*setColumnIndex*(iv, col);

grid.getChildren().addAll(iv);

**Tabellenansichten**

**Entwurf**

Damit Spieler die Historie betrachten können, wurden Tabellen implementiert, in die die alten Spielstände und Spielzüge hineingeladen werden.

Um dies zu realisieren, muss in dem Controller ein TableView erzeugt werden. Innerhalb des TableViews werden dann die verschiedenen Spalten definiert.



Da in die Tabelle die Datenbankinhalte geschrieben werden und diese bei jedem Historienaufruf frisch geladen werden, muss der Inhalt der Tabelle zwingend dynamisch sein.

Um diese Dynamik zu gewährleisten, wird das Füllen der Tabelle in dem Controller für diese FXML-Seite durchgeführt. Um dies zu erreichen, muss die Tabelle und die Spalten von der FXML-Seite in der Controllerklasse bekannt gemacht werden:

@FXML

TableView<ValueClass> tableView, tZuege;

@FXML

TableColumn<ValueClass, String> col1, col2, col3,col4, col5;

## Künstliche Intelligenz

Die Künstliche Intelligenz ist eine selbst erdachte Konzeptionierung. Ziel dieser ist es, in erster Linie Gewinnmuster zu erkennen und anschließend die dafür benötigte Spaltennummer auszugeben. Dabei wird zuerst darauf geachtet, ob der Spieler eine Siegmöglichkeit hat. Sollte dies nicht der Fall sein, wird überprüft, ob bei dem Gegenspieler eine Siegchance im nächsten Zug bestehen würde. Findet die KI zu diesen beiden genannten Fällen keine Lösung, wird aus performence-technischen Gründen eine Zufallszeile generiert.

Die Künstliche Intelligenz ist in einer einzigen Klasse (KiMain.java) programmiert, welche folgende Methoden bietet:

* public KiMain()

**Entwurf**

* public void setEigenerStein()
* public void setGegnerStein()
* public void berechne()
* public int get\_spalte()
* public int[] getletzter\_zug()

Die beiden Set-Klassen dienen dazu, um die gespielten Züge der beiden Agents in das eigene Spielfeld der KI-Klasse abzuspeichern.

**public** **void** setGegnerStein(**int** spalte) {

spalte\_rueckgabe = spalte;

feld [akt\_zeile[spalte]][spalte\_rueckgabe] = 2;

letzter\_zug[1] = spalte\_rueckgabe;

letzter\_zug[0] = akt\_zeile[spalte];

akt\_zeile[spalte]--;

}// end setter

In der Klasse *berechne()* befindet sich der im obigen Teil des Kapitels beschriebene Algorithmus, welcher den aktuellen Zug berechnet. Beispiel:

**try**{

**if**(feld[zeile+1][spalte+1] == 1 && feld[zeile+2][spalte+2] == 1 && feld[zeile+3][spalte+3] == 1){ //0111 unten rechts

**if** (zeile < 5 && feld[zeile+1][spalte] != 0){

ergebnis = spalte;

**break**;

} **else** **if** (zeile ==5 ){

ergebnis = spalte;

**break**;

}

} **catch** (ArrayIndexOutOfBoundsException e){}

*Get\_spalte()* ist eine Return-Funktion, die das berechnete Ergebnis an die jeweilige Serverschnittstelle weiterleitet.

Die Methode *getletzter\_zug()* ist auch eine Return-Funktion. Ihr Zweck ist es, die aktuellen X- und Y-Koordinaten des letzten Zuges an die graphische Oberfläche weiterzuleiten.

# Implementierung

**Implementierung**

## Java-API Dokumentation

Die vollständige Dokumentation der Implementierung wurde aus dieser technischen Dokumentation ausgegliedert und befindet sich in der beiliegenden Java API Dokumentation im Projektordner.

## (Sequenzdiagramm)

# Test

**Test**

## Testverfahren

Um das Programm „Claire’s Vier gewinnt“ zu testen, wurden die vier, in Abbildung **X** aufgezeigten, Testverfahren durchlaufen. Diese, auch in der einschlägigen Fachliteratur empfohlenen Testverfahren, wurden gewählt, um ein möglichst weitgehendes Testen auf allen Ebenen der Software vor der Abnahme zu gewährleisten.



**Abbildung 5: Durchgeführte Testverfahren**

Bei den **Modultests** wurden die einzelnen Module grafische Benutzeroberfläche, Serverkommunikation, künstliche Intelligenz und die Datenbank auf ihre Funktionalitäten getestet.

Bei den **Integrationstests** wurden jeweils die Schnittstellen zwischen den einzelnen Modulen hergestellt und getestet. Dies geschah zuerst zwischen der grafischen Benutzeroberfläche und der Serverkommunikation. Anschließend wurde das Modul künstliche Intelligenz in den Test mit einbezogen. Zum Schluss wurde die Datenbank integriert und getestet.

Für den **Systemtest** wurden in Kapitel **X.2** mehrere Testfälle in Form von Geschäftsanwendungsfällen geschrieben. Auf deren Basis erfolgte ein Test der gesamten Anwendung. Hierbei gab es einige Überschneidungen zwischen den Systemtest und dem Abnahmetest, in dem die Abdeckung der gegebenen Anforderungen getestet wurde.

Die aufgetretenen Fehler in den einzelnen Testphasen wurden nach dem Testen an den zuständigen Entwickler weitergeleitet und im weiteren Verlauf der Entwicklung behoben.

Der zuletzt durchgeführte **Abnahmetest** hat gezeigt, dass alle Minimal-Anforderungen des Auftraggebers erfüllt worden sind.

## Testfälle

**Test**

Dieses Kapitel enthält auszugsweise vier durchgeführte dokumentierte Testfälle.

* (1) Teilnahme an Turnier
* (2) Sieger des Satzes manuell bestimmen
* (3) Sieger des Spiels manuell bestimmen
* (4) Historie anzeigen

Für diese Testfälle wurde ein einheitliches Template verwendet. Alle Testfälle sind auf der Ebene des **Systemtests** einzuordnen.

|  |  |  |
| --- | --- | --- |
| Name | Teilnahme an Turnier | |
| Nummer | 1 | |
| Art | Testfall | |
| Kurzbeschreibung | Software Agent spielt autom. gegen anderen Software Agent | |
| Auslöser |  | |
| Ergebnis | Ein Satz wurde gegeneinander gespielt | |
| Akteure | Software Agents | |
| Eingehende Informationen | Keine | |
| Vorbedingungen | -Server ist gestartet  -Anwendung ist gestartet | |
| Nachbedingungen | Keine | |
|  | **Schritte** | **Ergebnis** |
| Ablauf | 1. Der User drückt Button „Spiel starten“. 2. Der User trägt Namen des Gegners ein. 3. Der User gibt die gewünschte Steinfarbe an. 4. Der User gibt die Serverschnittstelle an. 5. Der User drückt den Button „Starten“. | 1. Der User gelangt in den Spielmodus. 2. Der User hat den Gegnernamen bestimmt. 3. Der User ist entweder „O“ oder „X“. 4. Die Kommunikation erfolgt entweder über File-Schnittstelle oder Pusher-Schnittstelle. |

**Tabelle 9: Testfall 1 Teilnahme am Turnier**

|  |  |  |
| --- | --- | --- |
| Name | Sieger des Satzes manuell bestimmen | |
| Nummer | 2 | |
| Art | Testfall | |
| Kurzbeschreibung | Spieler bestimmt manuell Ausgang des Satzes | |
| Auslöser | Satz ist zu Ende gespielt | |
| Ergebnis | Der Ausgang des Satzes wurde geändert | |
| Akteure | Spieler | |
| Eingehende Informationen | Keine | |
| Vorbedingungen | Satz ist zu Ende | |
| Nachbedingungen | Keine | |
|  | **Schritte** | **Ergebnis** |
| Ablauf | 1. Der User drückt Button „Sieger ändern“. | 1. Auf dem Screen ändert sich der Wert „gewonnen“ zu „verloren“ bzw. umgekehrt. |

**Tabelle 10: Testfall 2 Sieger des Satzes manuell bestimmen**

**Test**

|  |  |  |
| --- | --- | --- |
| Name | Sieger des Spiels manuell bestimmen | |
| Nummer | 3 | |
| Art | Testfall | |
| Kurzbeschreibung | Spieler bestimmt manuell Ausgang des Spiels | |
| Auslöser | Spiel ist zu Ende gespielt | |
| Ergebnis | Die Punkte eines Spiels wurden geändert | |
| Akteure | Spieler | |
| Eingehende Informationen | Keine | |
| Vorbedingungen | Spiel ist zu Ende | |
| Nachbedingungen | Keine | |
|  | **Schritte** | **Ergebnis** |
| Ablauf | 1. Der User drückt in der Tabelle den Satz an, welchen er auf den anderen Wert („gewonnen“/“Verloren“) setzen möchte.  2. Der User drückt „Button“ beenden. | 1. Auf dem Screen ändert sich der Wert „gewonnen“ zu „verloren“ bzw. umgekehrt.  2. Der User gelangt zurück zum Startbildschirm. |

**Tabelle 11: Testfall 3 Sieger des Spieles manuell bestimmen**

|  |  |  |
| --- | --- | --- |
| Name | Historie anzeigen | |
| Nummer | 4 | |
| Art | Testfall | |
| Kurzbeschreibung | Anzeige der gespielten Spiele inklusive der zugehörigen Sätze und Züge | |
| Auslöser | Spieler möchte sich vergangene Spiele ansehen | |
| Ergebnis | Übersicht der Spieler | |
| Akteure | Spieler | |
| Eingehende Informationen | Keine | |
| Vorbedingungen | Anwendung ist gestartet | |
| Nachbedingungen | Keine | |
|  | **Schritte** | **Ergebnis** |
| Ablauf | 1. Der User drückt Button „Historie betrachten“. 2. Der User klickt auf ein Spiel. 3. Der User klickt auf einen Satz. | 1. Der User erhält eine Übersicht aller Spiele. 2. Der User erhält eine Übersicht der zugehörigen Sätze zum gewählten Spiel. 3. Der User sieht den endgültigen Spielstand des gewählten Satzes auf dem Spielfeld. |

**Tabelle 12: Testfall 4 Historie anzeigen**

**Test**

# Inbetriebnahme

**Inbetriebnahme**

Als Inbetriebnahme des Programms „Vier gewinnt“ wird, wie zu Beginn definiert, die erstmalige Teilnahme am gewerteten Turnier in der Vorlesung erachtet. Dort wird eine Partie gegen den bereitgestellten Server ausgetragen.

Eine Installation des Programmes ist nicht notwendig. Es genügt auf dem entsprechenden Rechner die Ausführung der JAR-Datei *viergewinnt.jar* zum Starten des Programmes. Es wird für weitere Erläuterungen zur Benutzung des Programmes auf das ebenfalls erstellte Benutzerhandbuch.pdf verwiesen.

# Anhang

**Anhang**

Anlage 1: Projektplan

1. <http://nadviser.de/Pg/Auftrag/Auftrag4gPush600.pdf> zuletzt abgerufen am 01.11.2015 17:25 Uhr [↑](#footnote-ref-1)
2. Vgl. Balzert, Helmut, Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering, 2009, S. 489 [↑](#footnote-ref-2)
3. Vgl. Balzert, Helmut, Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering, 2009, S. 456 [↑](#footnote-ref-3)