# 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 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Inhaltsverzeichnis

[Änderungshistorie](#_Toc431214766)

[Inhaltsverzeichnis I](#_Toc431214767)

[Abkürzungsverzeichnis I](#_Toc431214768)

[Abbildungsverzeichnis I](#_Toc431214769)

[Tabellenverzeichnis I](#_Toc431214770)

[Anlagenverzeichnis I](#_Toc431214771)

[1. Einleitung 1](#_Toc431214772)

[1.1 Projektziel 1](#_Toc431214773)

[1.2 Vorgehensmodell 1](#_Toc431214774)

[2. Projektplanung 1](#_Toc431214775)

[2.1 Definition der Arbeitspakete 1](#_Toc431214776)

[2.2 Projektplan 1](#_Toc431214777)

[3. Anforderungsanalyse 1](#_Toc431214778)

[3.1 Ermittlung der Projekt-Anforderungen 1](#_Toc431214779)

[3.1.1 Geschäftsanwendungsfälle 1](#_Toc431214780)

[3.1.2 Systemanwendungsfälle 1](#_Toc431214781)

[3.2 Spezifikation der Projekt-Anforderungen 1](#_Toc431214782)

[3.2.1 Funktionale Anforderungen 1](#_Toc431214783)

[3.2.2 Nichtfunktionale Anforderungen 1](#_Toc431214784)

[3.2.3 Rahmenbedingungen 1](#_Toc431214785)

[4. Entwurf 1](#_Toc431214786)

[4.1 Struktur des Systems 1](#_Toc431214787)

[4.1.1 Schichten-Architektur 1](#_Toc431214788)

[4.1.2 Klassendiagramme 1](#_Toc431214789)

[4.2 Verhalten des Systems 1](#_Toc431214790)

[4.2.1 Aktivitätsdiagramme 1](#_Toc431214791)

[4.2.2 Zustandsdiagramme 1](#_Toc431214792)

[4.3 Logische Komponenten des Systems 1](#_Toc431214793)

[4.3.1 Datenbank 1](#_Toc431214794)

[4.3.2 Server-Schnittstellen 1](#_Toc431214795)

[4.3.2.1 File-Schnittstelle 1](#_Toc431214796)

[4.3.2.2 Pusher-Schnittstelle 1](#_Toc431214797)

[4.3.3 Grafische Benutzeroberfläche 1](#_Toc431214798)

[4.3.4 Künstliche Intelligenz 1](#_Toc431214799)

[5. Implementierung 1](#_Toc431214800)

[5.1 Java-API Dokumentation 1](#_Toc431214801)

[5.2 (Sequenzdiagramm) 1](#_Toc431214802)

[6. Test 1](#_Toc431214803)

[6.1 Testverfahren 1](#_Toc431214804)

[6.2 (…) 1](#_Toc431214805)

[7. Abnahme und Inbetriebnahme 1](#_Toc431214806)

[Anhang 1](#_Toc431214807)

# Abkürzungsverzeichnis

**A**  
AO Abgabenordnung  
AktG Aktiengesetz

**B**  
BPMN Business Process Model and Notation  
BDSG Bundesdatenschutzgesetz

**C**  
CISO Chief Information Security Officer

**G**  
GmbHG GmbH-Gesetz

**I**ISP Internet Service Provider  
IEC International Electrotechnical Commission  
ISMS Informationssicherheitsmanagementsystem  
ISO International Organization for Standardization

**M**MTP Move-To-Production

**P**PERL Practical Extraction and Report Language

**S**  
SQL Structured Query Language

**U**UML Unified Modeling Language

**X**XML Extensible Markup Language

# Abbildungsverzeichnis

[**Abbildung 1: Tabellarische Auflistung der Arbeitspakete** 1](#_Toc431823122)

# Tabellenverzeichnis

[**Tabelle 1: Funktionale Anforderungen** 1](#_Toc431824845)

[**Tabelle 2: Nichtfunktionale Anforderungen** 1](#_Toc431824846)

# Anlagenverzeichnis

[Anlage 1: Stakeholder-Analyse 1](#_Toc431207063)

# 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.

## Vorgehensmodell

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. Diese sollen in Tabelle **X** kurz gegenübergestellt werden.

|  |  |  |
| --- | --- | --- |
| Wasserfallmodell | Scrum |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Tabelle 1: Gegenüberstellung Vorgehensmodelle**

Die finale Entscheidung fiel nach gemeinsamer Diskussion dann auf das Vorgehensmodell „Scrum“.

# Projektplanung

## Definition der Arbeitspakete

Zu Beginn des Projektes wurden nach intensiver Analyse des Projektauftrages alle vorgegebenen Anforderungen in Arbeitspakete übersetzt und dokumentiert. Bei diesen handelt es sich, wie in Abbildung **X** aufgezeigt, um:



**Abbildung 1: Tabellarische Auflistung der Arbeitspakete**

## Projektplan

Im nachfolgenden Schritt wurden alle in Kapitel 2.1 aufgeführten Arbeitspakete in einen ganzheitlichen Projektplan überführt. Wesentliches Ziel dieses Projektplanes ist die zeitliche Planung der Arbeitspakete unter Berücksichtigung der an den Abgabeterminen geforderten Komponenten, die gleichermaßen als *Meilensteine* im Projektplan erscheinen. Außerdem wird darin eine Zuweisung von Verantwortlichkeiten an die Projektteilnehmer getroffen. So ist es möglich, jederzeit die aktuell zu bearbeitende Aufgabe des jeweiligen Projektteilnehmers einsehen zu können.

Im Sinne des Vorgehensmodells „Scrum“ wurde der Projektplan vergleichsweise dynamisch gestaltet und rekursiv bei jedem gemeinsamen Treffen angepasst beziehungsweise korrigiert. Dies ist insbesondere auf sich häufig wechselnde Rahmenbedingungen innerhalb des Projektumfeldes zurückzuführen.

Der vollständige Projektplan ist aus Gründen der Übersichtlichkeit aus diesem Kapitel ausgegliedert und ist im Anhang der größeren Anlage **X** zu entnehmen.

# Anforderungsanalyse

## Ermittlung der Projekt-Anforderungen

## Geschäftsanwendungsfälle

## Systemanwendungsfälle

## 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.[[1]](#footnote-1)

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 2: Funktionale Anforderungen**

## Nichtfunktionale Anforderungen

Nach Helmut Balzert legt eine nichtfunktionale Anforderung „(…) eine vom Softwaresystem oder einer seiner Komponenten bereitzustellende Funktion oder bereitzustellenden Service fest“.[[2]](#footnote-2) 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 3: Nichtfunktionale Anforderungen**

## Rahmenbedingungen

# Entwurf

## System Architektur

Der 4-Gewinnt Agent Claire ist nach einer klassischen 3-Schichte-Architektur aufgebaut. Somit besteht die Architektur aus folgenden Bausteinen:

* Graphische Oberfläche
* Hauptapplikation
* Datenbank

(Hier Graphikeinfügen

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

Die Applikationsebene selber besteht aus 3 Teilen. Zum einen die künstliche Intelligenz, 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.

## Klassendiagramme

## Verhalten des Systems

## Aktivitätsdiagramme

## Zustandsdiagramme

## Logische Komponenten des Systems

## Datenbank

Die folgende Abbildung der Datenbankmodellierungzeigt anschaulich das Datenbankmodell der Anwendung.



**Abbildung 2: Datenbank Modellierung**

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 Packet Datenbank befindet sich eine Klasse für die Datenbank, welche den Zugriff auf die Datenbank mittels den SQL-Befehlen abbildet. Des Weiteren kümmern sich drei zusätzliche Klassen darum, dass auf die Daten des aktuellen Spielers, das aktuellen Spiels und des aktuellen Satz zugegriffen werden kann. Dies geschieht mittels Speicherung der Werte in die statischen Klassenvariablen.

## Server-Schnittstellen

## File-Schnittstelle

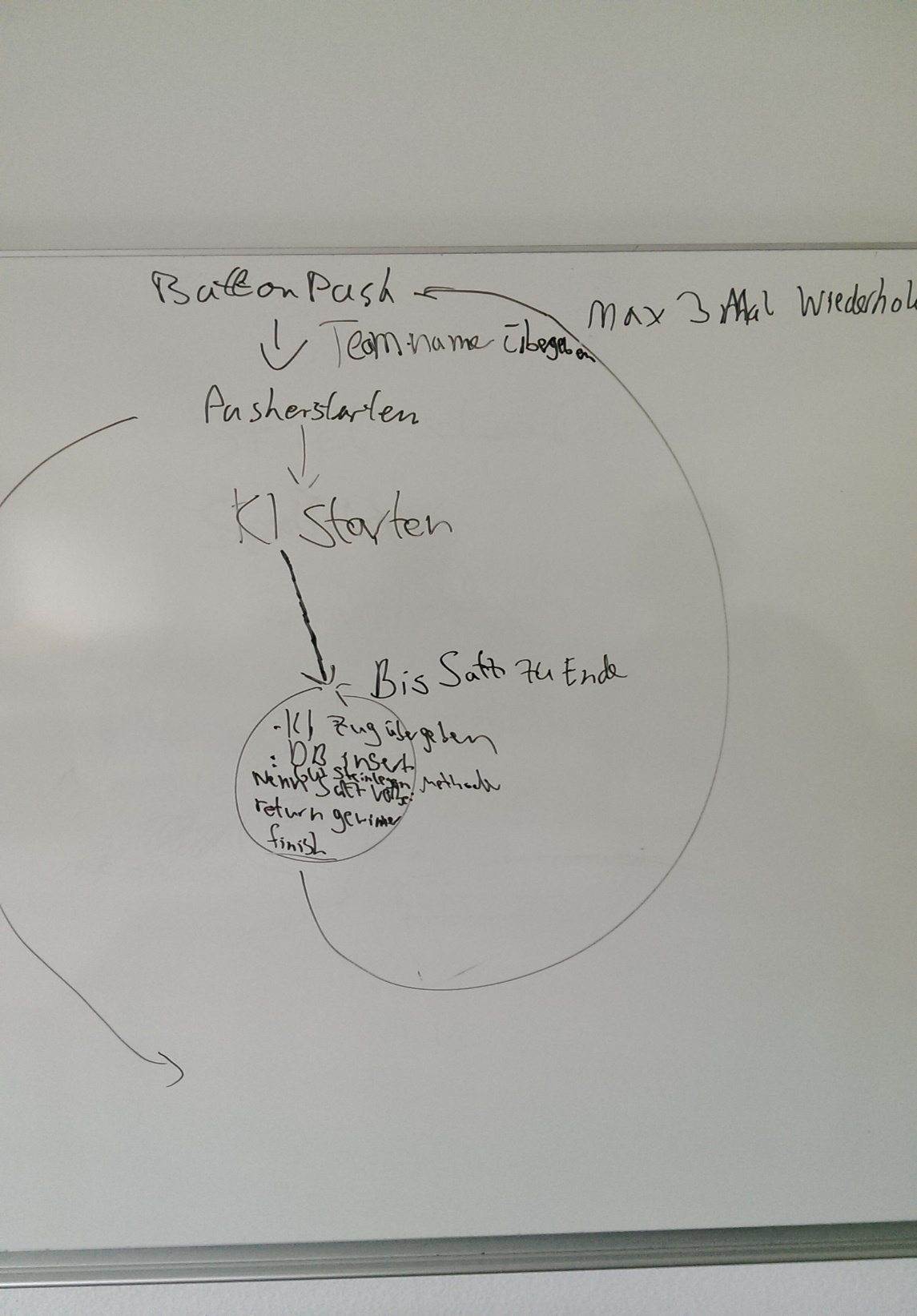
* Auswahl eines Ordners, der zur Kommunikation zwischen den beiden Clients und dem Server dient
* Auf Client wird Team (O oder X) ausgewählt
* Anhand des Team-Namens werden die Dateien zur Kommunikation der zwei Teams erstellt
* Server kann anhand dieser Dateien (deren Namen) feststellen, welches Team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

## 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()
* 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.

Bsp.:

**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

## Java-API Dokumentation

Die Dokumentation der Implementierung befindet sich in der Java API Dokumentation.

## (Sequenzdiagramm)

# Test

## Testverfahren

Um die das Programm „Claire“ zu testen, wurden folgende Testphasen durchlaufen:

* Modultest
* Integrationstest
* Systemtest
* Abnahmetest

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 künstlichen Intelligenz in den Test mit einbezogen. Zum Schluss wurde die Datenbank integriert und getestet.

Für den Systemtest wurde mehrere Testfälle in Form von Geschäftsanwendungsfällen geschrieben. Auf dessen 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 würden an den jeweiligen Entwickler weitergeleitet und behoben.

Der zuletzt durchgeführte Abnahmetest hat gezeigt, dass alle Anforderungen erfüllt wurden sind

## Testfälle

|  |  |  |
| --- | --- | --- |
| Name | Teilnahme an Tunier | |
| 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. User drückt Button „Spiel starten“ 2. User trägt Namen des Gegners ein 3. User gibt gewünschte Steinfarbe an 4. User gibt Serverschnittstelle an 5. User drückt den Button „Starten“ | 1. User gelangt in Spielmodus 2. Der Spieler hat den Gegnernamen bestimmt 3. Spieler ist entweder „O“ oder „X“ 4. Die Kommunikation erfolgt entweder über die File-Schnittstelle oder die Pusher-Schnittstelle |

|  |  |  |
| --- | --- | --- |
| 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 Augang des Satzes wurde geändert | |
| Akteure | Spieler | |
| Eingehende Informationen | Keine | |
| Vorbedingungen | Satz ist zu Ende | |
| Nachbedingungen | Keine | |
|  | **Schritte** | **Ergebnis** |
| Ablauf | 1. User drückt Button „Sieger ändern“ | 1. Auf dem Screen ändert sich der Wert „gewonnen“ zu „verloren“ bzw. umgekehrt |

|  |  |  |
| --- | --- | --- |
| 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. User drückt in der Tabelle den Satz an, welchen er auf den anderen Wert („gewonnen“/“Verloren“) setzen möchte  2. User drückt „Button“ beenden | 1. Auf dem Screen ändert sich der Wert „gewonnen“ zu „verloren“ bzw. umgekehrt  2. User gelangt zurück zum Startbildschirm |

|  |  |  |
| --- | --- | --- |
| 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. User drückt Button „Historie betrachten“ 2. User klickt auf ein Spiel 3. User klickt auf ein Satz | 1. User erhält eine Übersicht aller Spiele 2. User erhält eine Übersicht der zugehörigen Sätze zum gewählten Spiel 3. User sieht den endgültigen Spielstand des gewählten Satzes auf dem Spielfeld |

# Abnahme und Inbetriebnahme

# Anhang

Anlage 1: Stakeholder-Analyse

1. Vgl. Balzert, Helmut, Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering, 2009, S. 489 [↑](#footnote-ref-1)
2. Vgl. Balzert, Helmut, Lehrbuch der Softwaretechnik – Basiskonzepte und Requirements Engineering, 2009, S. 456 [↑](#footnote-ref-2)