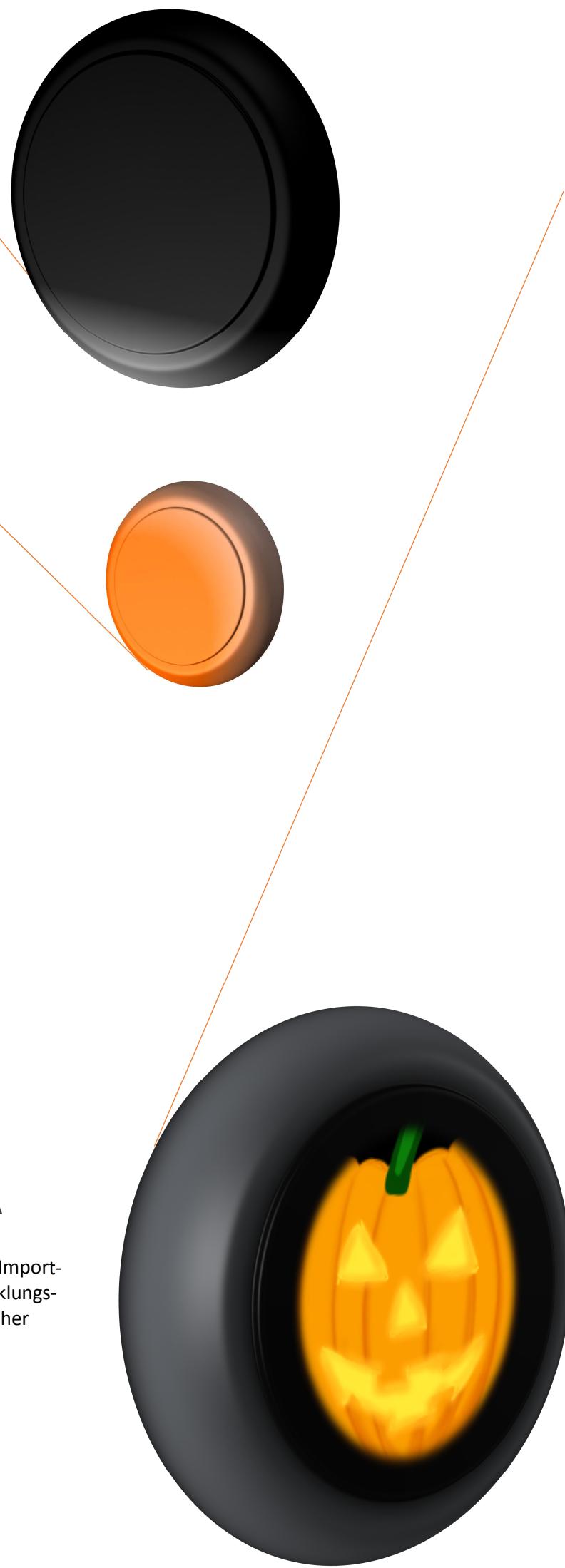


Dokumentation VIER.

Projekt WS 2016/2017, Kurs WWI14SCA

Die Dokumentation umfasst das Pflichtenheft, die Importanleitung, die Benutzerdokumentation, die Entwicklungsdokumentation sowie die Dokumentation zusätzlicher Funktionen zur programmierten Anwendung VIER.

**Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub
14.11.2016**





Inhalt

Inhalt.....	1
Abbildungsverzeichnis	3
Tabellenverzeichnis	4

Pflichtenheft 5

Inhalt.....	6
--------------------	----------

1. Zielbestimmung	7
1.1. Musskriterien	7
1.2. Wunschkriterien.....	8
1.3. Abgrenzungskriterien	8
2. Produkteinsatz.....	9
2.1. Anwendungsbereich.....	9
2.2. Zielgruppen.....	9
2.3. Betriebsbedingungen	9
3. Produktübersicht	10
4. Produktfunktion	11
4.1. Benutzerfunktionen.....	11
4.2. Agentfunktionen.....	12
5. Produktdaten.....	13
6. Produktleistung	14
7. Qualitätsanforderungen.....	14
8. Benutzeroberfläche.....	15
8.1. Dialogstruktur.....	15
8.2. Bildschirmlayout.....	16
9. Nichtfunktionale Anforderungen.....	17
10. Technische Produktumgebung	17
10.1. Software	17
10.2. Hardware	17
10.3. Orgware	17
10.4. Produktschnittstellen	17
11. Spezielle Anforderungen an die Entwicklungsumgebung	18
11.1. Software	18
11.2. Hardware	18



11.3.	Orgware	18
11.4.	Entwicklungsschnittstellen.....	18
12.	Ergänzungen	18
13.	Testfälle.....	19
14.	Glossar.....	20
15.	Referenzen	21

Importanleitung 22

1.	Einleitung:	23
2.	Systemvoraussetzungen:.....	23
3.	Projekt auf dem PC speichern.....	23
4.	Das Projekt nach Eclipse importieren	24
5.	Applikation starten	24

Benutzerdokumentation 25

Inhalt.....	26	
Abbildungsverzeichnis	27	
1.	Allgemeine Problemstellung	28
2.	Spielregeln.....	28
3.	Ablaufbedingungen.....	29
4.	Installation des Programms.....	29
5.	Start der Software.....	30
6.	Anmeldung der Spieler.....	30
7.	Aufbau des Hauptbildschirms.....	31
8.	Konfiguration des Spiels.....	34
8.1.	Manuelles Spiel	34
8.2.	Automatisches Spiel	35
9.	Spiel starten.....	37
9.1.	Manuelles Spiel	37
9.2.	Automatisches Spiel	38
10.	Nachverfolgen vergangener Spiele	39
11.	Verlassen des Spiels	40
12.	Fehlermeldungen.....	40
12.1.	Fehlerhafte Eingabe der Spielernamen.....	40
12.1.1.	Eingabe keines bzw. nur eines Spielernamens	40



12.1.2.	Eingabe eines identischen Spielernamens.....	41
12.2.	Fehlender Kontaktpfad bei Konfiguration der File-Schnittstelle	41
12.3.	Auswahl des Spielmodus „Gegen den Computer“.....	41
12.4.	Der Pusher-Server reagiert nicht.....	42
12.5.	Der Server reagiert nicht	42
13.	Mit dem Agenten im Turnier spielen	42

Entwicklungsdocumentation 46

Inhalt	47
---------------------	-----------

Abbildungsverzeichnis	49
------------------------------------	-----------

1. Einführung	50
----------------------------	-----------

1.1.	Was ist VIER.?	50
1.2.	Wesentliche Features.....	50
1.3.	Qualitätsziele.....	50
1.4.	Stakeholder	51

2. Randbedingungen	51
---------------------------------	-----------

2.1.	Technische Randbedingungen	51
2.2.	Organisatorische Randbedingungen	52
2.3.	Konventionen	52

3. Fachlicher Kontext	53
------------------------------------	-----------

3.1.	Menschlicher Gegner (Benutzer)	53
3.2.	KI Gegner (Fremdsystem)	53

4. High System Level Overview	54
--	-----------

4.1.	Logik.....	54
4.2.	Schnittstellen.....	54
4.3.	Graphische Oberfläche	55
4.4.	Datenhaltung.....	55

5. Produktstruktur	55
---------------------------------	-----------

6. Fachklassen	56
-----------------------------	-----------

7. Konzeption	58
----------------------------	-----------

7.1.	Die Startklasse	58
7.2.	Die Spiellogik	59
7.2.1.	Der Algorithmus – Klasse „AlphaBeta“.....	59
7.2.2.	Das Spielobjekt – Klasse „Game“.....	62
7.2.3.	Das Satzobjekt – Klasse „Match“.....	63
7.2.4.	Die Schnittstelle – Klasse „NameListener“	65



7.2.5.	Das Spielerobjekt – Klasse „Player“	65
7.2.6.	Eine Spielposition – Klasse „PlaySlot“	66
7.2.7.	Der Spielzug – Klasse „Turn“.....	66
7.3.	Schnittstellen	67
7.3.1.	Klasse PusherInterface.java	68
7.3.2.	Klasse FileInterface.java	69
7.3.3.	Interfaces im Package Interfaces.....	70
7.4.	Gui	70
7.4.1.	Stages.....	71
7.4.2.	Layout	72
7.4.3.	EventHandler	73
7.4.4.	Das Spielfeld	75
7.4.5.	Das Menü.....	77
7.5.	Datenbank	78
7.5.1.	ConnectHSQL.java.....	79
7.5.2.	DBConnector.java	80
7.5.3.	Getter – Methoden .java	80
7.5.4.	Setter – Methoden .java	81
7.5.5.	LegacyMethods.java	81
7.5.6.	Datenbank Aufbau	81
8.	Programmablauf	82
8.1.	Satz spielen	82
8.2.	Gespielte Spiele nachverfolgen.....	84
9.	JavaDoc	85
10.	Referenz	85
Extras	86
Extras	87



Abbildungsverzeichnis

Abbildung 1: Anwendungsfalldiagramm	10
Abbildung 2: Dialogstruktur der Hauptseite	15
Abbildung 3: Hauptbildschirm.....	16
Abbildung 4: Reiter Optionen.....	16
Abbildung 5: Reiter Themen.....	16
Abbildung 6: Archivdatei auf den PC runterladen.....	23
Abbildung 7: Gewinnsituation "4 gewinnt"	28
Abbildung 8: Anmeldemaske	30
Abbildung 9: Aufbau des Hauptbildschirms	31
Abbildung 10: Reiter Optionen.....	32
Abbildung 11: Reiter Themen	32
Abbildung 12: Hauptbildschirm in Thema Süßigkeiten.....	32
Abbildung 13: Hauptbildschirm in Thema Food.....	33
Abbildung 14: Hauptbildschirm in Thema Sports.....	33
Abbildung 15: Warnmeldung bei Themenwechsel im laufenden Spiel	34
Abbildung 16: Auswahl des Spielmodus.....	34
Abbildung 17: Einstellungen für ein automatisches Spiel	35
Abbildung 18: Angabe des Kontaktpfads	36
Abbildung 19: Einstellungsoptionen der Pusher-Schnittstelle.....	37
Abbildung 20: Manuelles Spiel starten	38
Abbildung 21: Vergangenes Spiel wieder aufnehmen	39
Abbildung 22: Nachverfolgen vergangener Spiele.....	39
Abbildung 23: Warnmeldung Spiel verlassen.....	40
Abbildung 24: Fehlermeldung Eingabe Spielernamen 1	40
Abbildung 25: Fehlermeldung Eingabe Spielernamen 2	41
Abbildung 26: Fehlermeldung fehlender Kontaktpfad	41
Abbildung 27: Fehlermeldung Spielmodus	41
Abbildung 28: Fehlermeldung Pusher-Server	42
Abbildung 29: Sequenzdiagramm Satzbeginn.....	43
Abbildung 30: Sequenzdiagramm Spielzug	44
Abbildung 31: Sequenzdiagramm Satzende.....	45
Abbildung 32: Fachlicher Kontext	53
Abbildung 33: High System Level Overview	54
Abbildung 34: Produktstruktur VIER.	56
Abbildung 35: Fachklassendiagramm VIER.	57
Abbildung 36: Ausschnitt eines Spielbaums von Tic Tac Toe	60
Abbildung 37: Beispiel für einen Alpha-Beta-Cut.....	61
Abbildung 38: Beispiel Berechnung isRow().....	64
Abbildung 39: Aufbau der GUI	73
Abbildung 40: Auszug Coding ChangeListener	74
Abbildung 41: Auszug Coding EventHandler	74
Abbildung 42: Auszug Coding LogIn	75
Abbildung 43: Auszug Coding GridPane	76
Abbildung 44: Auszug Coding Methoden auf GridPane	77
Abbildung 45: Aufbau Menü	78
Abbildung 46: ER-Diagramm der Entitäten	81
Abbildung 47: Aktivitätendiagramm "Satz spielen"	83
Abbildung 48: Aktivitätendiagramm "Spiel nachverfolgen"	84



Tabellenverzeichnis

Tabelle 1: Übersicht über die Qualitätsanforderungen	14
Tabelle 2: Qualitätsziele	50
Tabelle 3: Stakeholder.....	51
Tabelle 4: Technische Randbedingungen.....	51
Tabelle 5: Organisatorische Randbedingungen	52
Tabelle 6: Konventionen	52



Pflichtenheft

VIER



(WWI14SCA, Praxisprojekt 2016)

Projekt: **VIER.**

Auftraggeber: **DHBW Mannheim**

Auftragnehmer: **WWI14SCA- Team VIER**

Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub

Version	Datum	Autor	Kommentar
0.1	26.09.2016	A. Neumann	Neu angelegt
0.2	03.10.2016	A. Neumann	Geschäftsprozesse und Anforderungen eingefügt
0.3	05.10.2016	A. Neumann	Glossar ergänzt
0.4	10.10.2016	L. Birk	Qualitätsanforderungen angepasst
0.5	10.10.2016	A. Neumann	Logo eingefügt, Benutzeroberfläche ergänzt
0.6	17.10.2016	A. Neumann	Ergänzung von Testfällen, Benutzeroberfläche angepasst
0.7	19.10.2016	A. Neumann	Glossar angepasst, Benutzeroberfläche finalisiert
0.8	11.11.2016	A. Neumann	Design Changes



Inhalt

Inhalt	6
1. Zielbestimmung	7
1.1. Musskriterien	7
1.2. Wunschkriterien	8
1.3. Abgrenzungskriterien	8
2. Produkteinsatz	9
2.1. Anwendungsbereich	9
2.2. Zielgruppen	9
2.3. Betriebsbedingungen	9
3. Produktübersicht	10
4. Produktfunktion	11
4.1. Benutzerfunktionen	11
4.2. Agentfunktionen	12
5. Produktdaten	13
6. Produktleistung	14
7. Qualitätsanforderungen	14
8. Benutzeroberfläche	15
8.1. Dialogstruktur	15
8.2. Bildschirmlayout	16
9. Nichtfunktionale Anforderungen	17
10. Technische Produktumgebung	17
10.1. Software	17
10.2. Hardware	17
10.3. Orgware	17
10.4. Produktschnittstellen	17
11. Spezielle Anforderungen an die Entwicklungsumgebung	18
11.1. Software	18
11.2. Hardware	18
11.3. Orgware	18
11.4. Entwicklungsschnittstellen	18
12. Ergänzungen	18
13. Testfälle	19
14. Glossar	20
15. Referenzen	21



1. Zielbestimmung

Es ist ein Software-Agent zu entwickeln, der das Spiel „4 gewinnt“ autonom gegen einen anderen Spieler spielen kann. Die Kommunikation beider Spieler soll dabei über einen zwischengeschalteten Server stattfinden, mit dem beide Spieler verbunden sind.

Die Spielzüge des Agenten sowie des Gegners sollen über ein graphisches Spielfeld nachverfolgt werden können. Spielzüge sowie Punkte sollen automatisch in einer Datenbank abgelegt werden und können durch den User beliebig aufgerufen und angezeigt werden.

1.1. Musskriterien

Agent:

- Der Agent muss eigenständig ohne Userinteraktion das Spiel „4 gewinnt“ spielen können
- Die Programmierung ist durch Java SE 8 und JavaFX zu realisieren
- Die Kommunikation zwischen Agent und Server muss über zwei Schnittstellen (Datei- sowie Pusher-Schnittstelle) stattfinden können
- Ein Wechsel zwischen den beiden Schnittstellen soll zwischen den Sätzen möglich sein
- Die Datei-Schnittstelle muss über Dateien auf Basis von Streams mit dem Server kommunizieren können
- Die Pusher-Schnittstelle muss über Events auf Basis von Websockets mit dem Server kommunizieren können
- Der Agent muss auf die Zugfreigabe durch den Server warten
- Der nächste Zug des Agenten ist autonom durch diesen zu berechnen und an den Server über die entsprechende Schnittstelle zu übermitteln

Spiel:

- Es muss ein fehlerfreier Ablauf eines Satzes eines Spiels gespielt werden können

Graphische Oberfläche:

- Es ist eine graphische Oberfläche zu gestalten, die den Satzstatus, den Spielstand sowie das Spielfeld anzeigt



- Auf der graphischen Oberfläche sollen alle Züge des aktuellen Satzes dargestellt werden

Datenhaltung:

- In der Datenbank HSQLDB sind die Daten Gegner, Startspieler, Sieger, Punkte, Spiele, Sätze und Züge kontinuierlich zu speichern
- Es müssen drei Abfragevarianten implementiert werden, die eine Nachverfolgung des Spielverlaufs eines beliebigen Spiels ermöglichen

1.2. Wunschkriterien

- Der Spielzug des Agenten muss in einer manuell einzustellenden Zugzeit errechnet und dem Server übermittelt werden, um dem Timeout des Servers zu entgehen
- Es kann mehr als ein Satz eines Spiels gespielt werden
- Es kann ausgewählt werden, ob der User manuell gegen einen Gegner oder der Agent autonom das Spiel spielt
- Der Agent erkennt eigenständig, wer der Gewinner ist
- Bei einem vorzeitigen Beenden des Servers, das zu einem abgebrochenen Satz führt, kann der Agent ein nachträgliches Editieren von Ergebnissen ermöglichen. Dies geschieht, indem alle eigenen und gelesenen Züge bis zum Abbruch automatisch gespeichert werden, das Spiel allerdings fortgeführt werden kann.

1.3. Abgrenzungskriterien

- Das Spiel ist vorerst nur in deutscher Sprache erhältlich
- Eine Steuerung über die Konsole ist nicht vorgesehen
- Die Kommunikation zwischen beiden Mitspielern findet nicht direkt, sondern immer nur über den Server statt



2. Produkteinsatz

Das Produkt dient zum autonomen Spiel des Spiels „4 gewinnt“ gegen andere Mitspieler.

2.1. Anwendungsbereich

Die Anwendung soll im Rahmen eines Turniers gegen Anwendungen anderer Entwickler im Spiel „4 gewinnt“ antreten. Sie wird auf einem Rechner im Computerlabor der DHBW Mannheim installiert. Eine kommerzielle Verwendung ist vorerst nicht vorgesehen.

2.2. Zielgruppen

Zielgruppe der Anwendung sind der Dozent sowie die Studenten des Kurses WWI14SCA. Dem Dozenten soll es dabei möglich sein die Anwendung eigenständig zu bedienen sowie zu installieren, um die Anwendung bewerten zu können.

Unerfahrene Benutzer erhalten über eine mitgelieferte Dokumentation, die eine Übersicht über den Aufbau sowie die Pflege der Anwendung liefert sowie einer Benutzeranleitung die Möglichkeit, die Anwendung zu bedienen.

2.3. Betriebsbedingungen

Die Anwendung soll auf jedem beliebigen Desktoprechner des Computerlabors der DHBW Mannheim lauffähig sein. Voraussetzung ist dabei, dass sie entsprechend der mitgelieferten Importanleitung installiert wurde.



3. Produktübersicht

Das folgende Anwendungsfalldiagramm (Abbildung 1) stellt die Funktionen sowie Akteure der Anwendung dar.

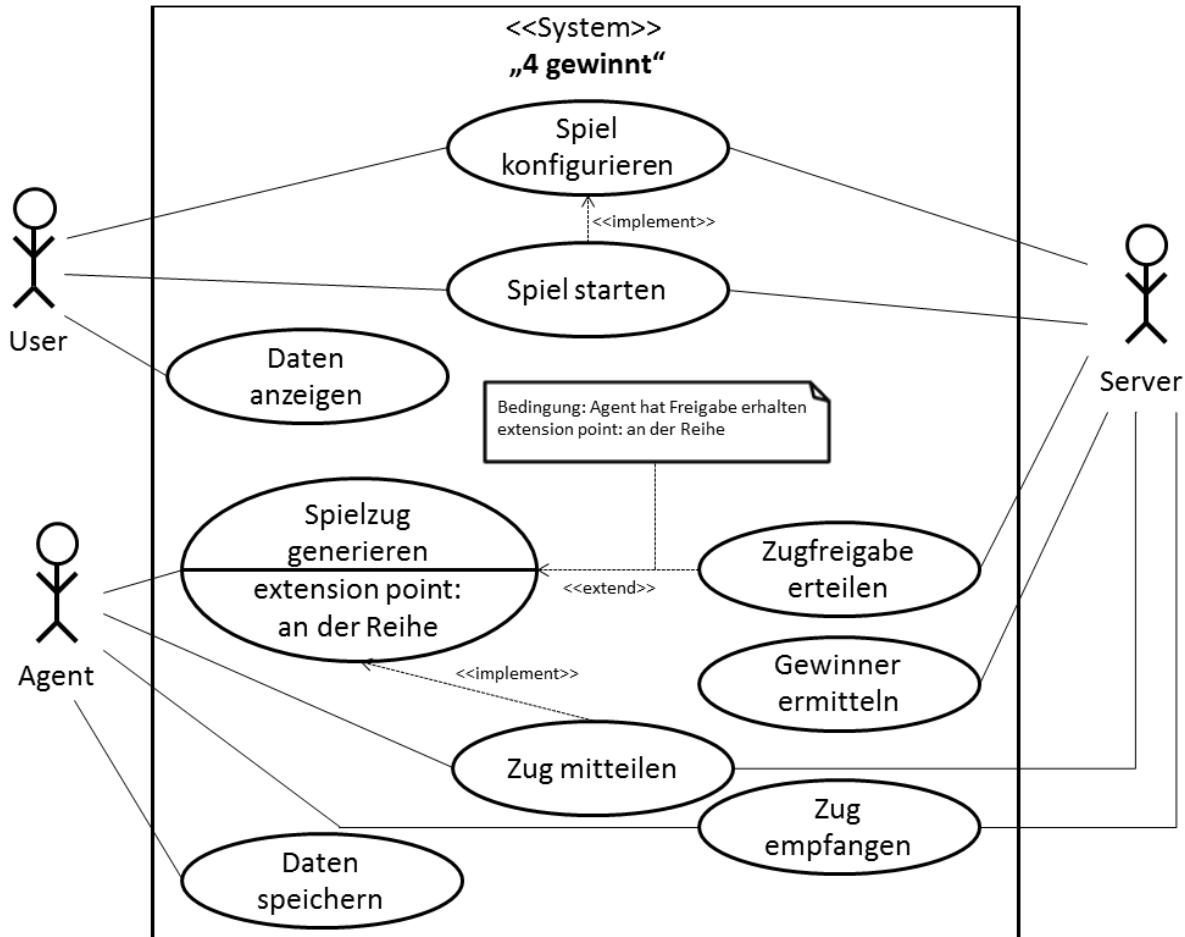


Abbildung 1: Anwendungsfalldiagramm

Es gibt in dem System „4 gewinnt“ drei Akteure – den User, den Agenten sowie den Server. Da hier lediglich die eigens entwickelte Anwendung dargestellt wird, ist der gegnerische Agent nicht aufgeführt.

Für eine weitere Übersichtlichkeit wurden ebenfalls die Datenbank sowie die graphische Oberfläche als Akteure nicht aufgeführt.



4. Produktfunktion

Es ist anzumerken, dass der Server bereits implementiert wurde und über eine der beiden Schnittstellen kontaktiert wird. Sämtliche Funktionen des Servers sind demnach bereits vorhanden und sollen daher an dieser Stelle keine weitere Erwähnung finden.

4.1. Benutzerfunktionen

/F100/

Geschäftsprozess:	Konfiguration des Spiels
Vorbedingung:	Anwendung ist auf dem Rechner installiert und gestartet
Nachbedingung Erfolg:	User kann das Spiel starten
Nachbedingung Fehlschlag:	User kann das Spiel nicht starten
Akteure:	User
Auslösendes Ereignis:	Anwendung wurde gestartet
Beschreibung:	Eingabe der Zugzeit in das entsprechende Eingabefenster

/F110/

Geschäftsprozess:	Spiel starten
Vorbedingung:	Das Spiel ist konfiguriert (/F100/)
Nachbedingung Erfolg:	Warten auf die Zugfreigabe durch den Server
Nachbedingung Fehlschlag:	Spiel startet nicht
Akteure:	User
Auslösendes Ereignis:	User drückt auf „Spiel starten“
Beschreibung:	User drückt auf „Spiel starten“, Agent wartet auf Spielfreigabe durch den Server

/F120/

Geschäftsprozess:	Daten anzeigen
Vorbedingung:	Der Agent hat bereits Spiele gespielt (/F110/)
Nachbedingung Erfolg:	gewünschte Daten werden angezeigt
Nachbedingung Fehlschlag:	es werden keine Daten angezeigt
Akteure:	User, Datenbank
Auslösendes Ereignis:	User gibt an, welches vergangene Spiel er nachverfolgen will
Beschreibung:	User gibt ein, welches Spiel er nachverfolgen will, Suche in der Datenbank, Datenbankabfrage wird auf der graphischen Oberfläche dargestellt

**/F130/**

Geschäftsprozess:	Thema der graphischen Oberfläche ändern
Vorbedingung:	Der Agent wurde geöffnet
Nachbedingung Erfolg:	Die graphische Oberfläche wird angepasst
Nachbedingung Fehlschlag:	die graphische Oberfläche ändert sich nicht
Akteure:	User, graphische Oberfläche, Agent
Auslösendes Ereignis:	User gibt an, welches graphische Thema er sehen möchte
Beschreibung:	User gibt ein, welche Oberfläche er sehen möchte, Spielfeld sowie Spielsteine werden entsprechend angepasst

4.2. Agentfunktionen

/F200/

Geschäftsprozess:	Spielzug generieren
Vorbedingung:	Spiel ist gestartet (/F110/), Agent ist an der Reihe, Gegnerischer Zug wurde empfangen (/F220/)
Nachbedingung Erfolg:	Zug ist berechnet und wird dem Server mitgeteilt (/F210/)
Nachbedingung Fehlschlag:	Time-Out durch den Server
Akteure:	Agent
Auslösendes Ereignis:	Zugfreigabe durch den Server
Beschreibung:	Auf Basis der implementierten Logik und vorheriger Züge wird neuer möglicher Spielzug generiert

/F210/

Geschäftsprozess:	Zug mitteilen
Vorbedingung:	Der Spielzug wurde berechnet (/F200/)
Nachbedingung Erfolg:	Warten auf die Antwort des Servers, Speicherung der Daten (/F230/)
Nachbedingung Fehlschlag:	Time-Out durch den Server
Akteure:	Agent
Auslösendes Ereignis:	Zug wurde generiert (/F200/)
Beschreibung:	Auf Basis der entsprechenden Schnittstelle wird die Information über den nächsten Zug an den Server übermittelt

/F220/

Geschäftsprozess:	Zug empfangen
Vorbedingung:	Der Gegner hat gezogen
Nachbedingung Erfolg:	Der nächste Zug kann berechnet werden (/F200/)
Nachbedingung Fehlschlag:	Time-Out durch den Server
Akteure:	Agent, Server
Auslösendes Ereignis:	Server sendet über entsprechende Schnittstelle den gegnerischen Zug
Beschreibung:	Je nach Schnittstelle wird der Zug des Gegners ausgelesen

**/F230/**

Geschäftsprozess:	Daten speichern
Vorbedingung:	Zug an Server übermittelt (/F210/)
Nachbedingung Erfolg:	Daten werden in der HSQLDB abgelegt
Nachbedingung Fehlschlag:	Daten können nicht gespeichert werden
Akteure:	Agent, Datenbank
Auslösendes Ereignis:	Agent sendet Zug an den Server und ist nicht an der Reihe
Beschreibung:	Daten des Spiels werden in der Datenbank abgelegt

/F240/

Geschäftsprozess:	Spielzüge graphisch anzeigen
Vorbedingung:	Zug an Server übermittelt (/F210/) bzw. gegnerischen Zug empfangen (/F220/) bzw. Daten werden angezeigt (/F120/)
Nachbedingung Erfolg:	Spielsteine werden in der graphischen Oberfläche angezeigt
Nachbedingung Fehlschlag:	Spielsteine werden nicht korrekt positioniert
Akteure:	Agent, graphische Oberfläche, Datenbank
Auslösendes Ereignis:	Agent sendet Zug an den Server bzw. Agent empfängt gegnerischen Zug durch den Server oder Daten eines vergangenen Spiels werden aufgerufen
Beschreibung:	Spielsteine werden an der korrekten Position im graphischen Spielfeld angezeigt

5. Produktdaten

Es sind folgende Daten persistent zu speichern:

/D10/ Spieldaten

Spiel-ID, Spieler1, Spieler2, Gewinner

/D20/ Satzdaten

Satz-ID, Satznummer, Spiel-ID, Punktestand

/D30/ Zugdaten

Zug-ID, Satz-ID, Spieler, Zeile, Spalte



6. Produktleistung

- /L10/ Die Zugzeit des Agenten auf den Gegnerzug darf maximal 2 Sekunden betragen, kann aber niedriger bis auf 0,5 Sekunden eingestellt werden.
- /L20/ Das Laden vorangegangener Spiele darf nicht länger als 20 Sekunden dauern
- /L30/ Die Startkonfiguration darf nicht länger als eine Minute dauern, da sonst die Userfreundlichkeit signifikant leidet.
- /L40/ Es müssen 0,3 Sekunden zwischen den Zugriffen des Agenten auf den Kontaktspfad liegen

7. Qualitätsanforderungen

Tabelle 1: Übersicht über die Qualitätsanforderungen

Kriterium	hoch relevant	relevant	nicht zwingend	nicht relevant
Funktionalität				
Angemessenheit	x			
Richtigkeit	x			
Ordnungsmäßigkeit		x		
Sicherheit				x
Zuverlässigkeit				
Reife		x		
Fehlertoleranz		x		
Wiederherstellbarkeit		x		
Benutzbarkeit				
Verständlichkeit	x			
Erlernbarkeit			x	
Bedienbarkeit	x			
Effizienz				
Zeitverhalten	x			
Verbrauchsverhalten		x		
Änderbarkeit				
Analysierbarkeit				x
Modifizierbarkeit		x		
Stabilität		x		
Prüfbarkeit		x		
Übertragbarkeit				
Anpassbarkeit	x			
Installierbarkeit	x			
Austauschbarkeit				x



8. Benutzeroberfläche

An dieser Stelle sei ein erster Oberflächenprototyp mit zugehöriger Dialogstruktur dargestellt.

8.1. Dialogstruktur

Im Folgenden wird die grobe Dialogstruktur einer fehlerfreien bzw. konfliktfreien Benutzung des Systems gezeigt.

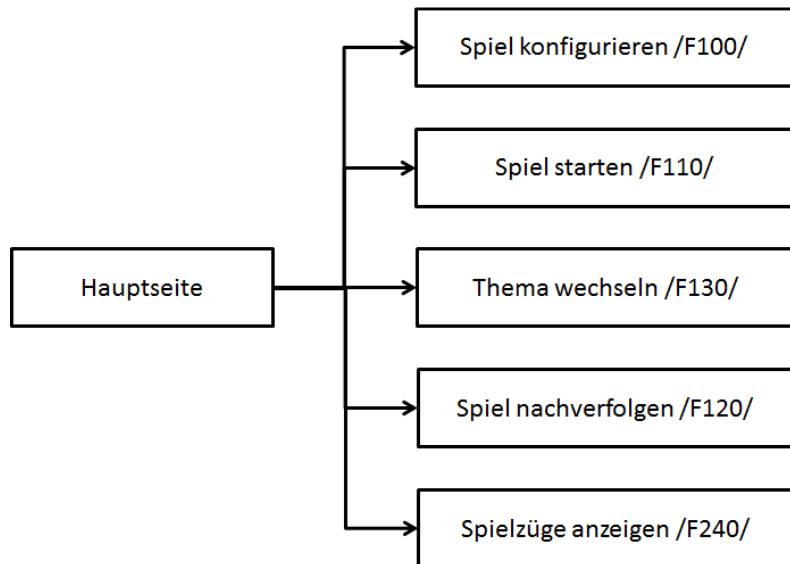


Abbildung 2: Dialogstruktur der Hauptseite

Die Hauptseite ist dabei die Seite, die nach Start des Agenten angezeigt wird. Auf der Hauptseite ist es dem User möglich das Spiel zu konfigurieren sowie ein Spiel zu starten. Er kann außerdem aus einer Menge von vier verschiedenen graphischen Oberflächen wählen. Es ist zudem möglich das Spiel in Echtzeit auf der graphischen Oberfläche anhand des dargestellten Spielfelds zu verfolgen. Ferner kann sich der User eine Liste der vergangenen Spiele anzeigen lassen und anschließend die Spielzüge eines dieser vergangenen Spiele nachverfolgen.



8.2. Bildschirmlayout

Es soll ein erster Prototyp der Benutzeroberfläche dargestellt werden, in dem sich die in Kapitel 8.1 erwähnte Dialogstruktur wiederspiegelt.

Der Hauptbildschirm ist durch Abbildung 3 dargestellt. Neben dem angezeigten Spielfeld können hier erste Konfigurationen durchgeführt werden. Es werden zudem der Spielstand sowie der Satzstatus angezeigt. Auf dem Spielfeld ist anschließend in Echtzeit das Setzen der Chips zu verfolgen.

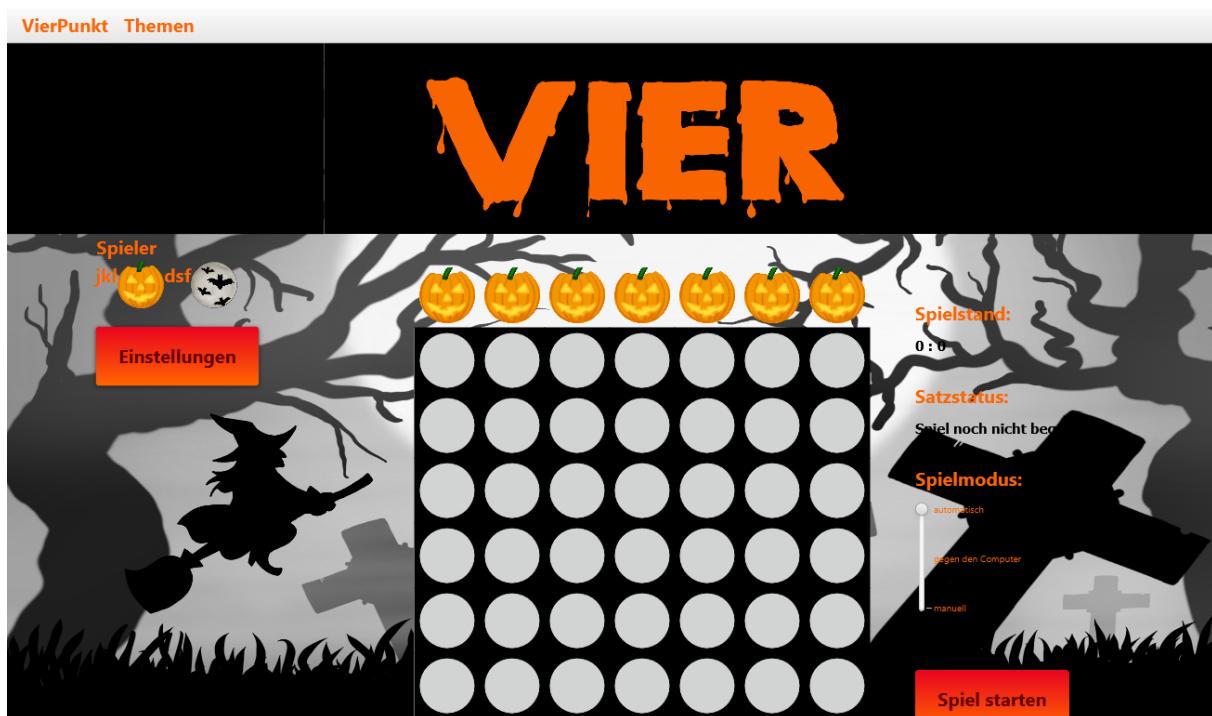


Abbildung 3: Hauptbildschirm

Der Hauptbildschirm inkorporiert zusätzlich ein Menüband mit Reitern, in denen weitere Funktionen eingebettet sind (Abbildung 4-5). Unter dem Reiter „Themen“ lässt sich dabei die graphische Oberfläche des Spielfeldes anpassen, während der Reiter „Optionen“ die Möglichkeit ein neues Spiel zu starten sowie bereits gespielte Spiele nachzuverfolgen bietet.



Abbildung 4: Reiter Optionen



Abbildung 5: Reiter Themen



9. Nichtfunktionale Anforderungen

- Das Regelwerk des Spiels „4 gewinnt“ ist zu beachten
- Der Agent darf den ordnungsgemäßen Serverbetrieb sowie den Betrieb des gegnerischen Agenten nicht beeinträchtigen/beeinflussen.
- Der Agent soll auf den Plattformen Windows sowie Mac OS laufen

10. Technische Produktumgebung

10.1. Software

- Betriebssystem Windows (ab Windows 7) oder Mac OS (ab Mac OS X)
- Laufzeitumgebung ab Java SE 7 (dementsprechend mit laufendem JavaFX-Framework)
- Datenbank HSQLDB (Version 2.3.4)
- Eclipse Neon

10.2. Hardware

- Herkömmlicher PC (z.B. Pentium 2 266 MHz oder schnellerer Prozessor mit einem physikalischen RAM von mindestens 128 MB)
- Maus & Tastatur
- Monitor

10.3. Orgware

- Netzwerkverbindung, damit über die Pusher- oder die Datei-Schnittstelle mit dem Server interagiert werden kann.

10.4. Produktschnittstellen

Es existieren keine Produktschnittstellen.



11. Spezielle Anforderungen an die Entwicklungsumgebung

11.1. Software

- Betriebssystem Windows (ab Windows 7) oder Mac OS (ab Mac OS X)
- Laufzeitumgebung ab Java SE 7 (dementsprechend mit laufendem JavaFx-Framework)
- Datenbank HSQLDB (Version 2.3.4)
- Eclipse Neon
- Zum Editieren und Überprüfen der erzeugten ASCII-Text-Dateien durch Kommunikation über die Datei-Schnittstelle ist der Windows-Editor ausreichend.

11.2. Hardware

siehe 10.2 Hardware

11.3. Orgware

Siehe 10.3 Orgware

11.4. Entwicklungsschnittstellen

keine

12. Ergänzungen

Die Anwendung soll zunächst in einer Basisversion erscheinen, die die hier spezifizierten Funktionen bedient.

Weitere Erweiterungen wie eine englische Sprachausgabe könnten in Version 2 inkorporiert werden.

Um die Anzeige bereits gespielter Spiele ausreichend nachvollziehen zu können, wird die Applikation zusammen mit Testdaten dreier Spiele ausgeliefert.



13. Testfälle

Um eine qualitativ hochwertige Anwendung abliefern zu können, sind folgende Testfälle vor Abgabe zu realisieren:

- /T010/** Der Agent spielt ein Spiel gegen einen anderen Agenten (/F110/, /F200/, /F210/, /F220/, /F230/, /F240/)
- /T020/** Der User verfolgt die Züge eines beliebigen vergangenen Spiels (/F120/, /F240/)
- /T030/** Der User lässt sich alle gespielten Spiele anzeigen (/F120/)
- /T040/** Der Server stürzt während des Spiels ab
- /T050/** Das Spiel wird vorzeitig abgebrochen
- /T060/** Der User ändert das Thema der graphischen Oberfläche (/F130/)
- /T070/** Ein abgebrochene Spiel wird nachverfolgt (/F120/, /F240/)
- /T080/** Die Schnittstelle wird zwischen den Sätzen gewechselt (/F210/, /F220/)
- /T090/** Es wird ein ganzes Spiel (alle drei Sätze) gespielt (/F110/, /F200/, /F210/, /F220/, /F230/, /F240/)



14. Glossar

Agent:

- Anwendung, die autonom als Spieler am Spiel „4 gewinnt“ teilnimmt
- Berechnet die Züge eigenständig und teilt sie dem Server mit

Datei-Schnittstelle:

- Kommunikation von Server und Agent über Dateien auf der Basis von Streams
- Der Zug des Agenten wird in einem definierten Pfad als Textdatei abgelegt

Hauptseite:

- Wird nach Start des Agenten angezeigt
- Erlaubt die Konfiguration sowie das Starten des Spiels
- Auf der Hauptseite lässt sich das Spiel anhand eines Spielfelds verfolgen

HSQLDB:

- Hyper Structured Query Language Database
- Freie und vollständig in Java programmierte relationale SQL-Datenbank
- HSQL lässt sich als eingebettetes Datenbanksystem in andere Applikationen integrieren

Pusher-Schnittstelle:

- Kommunikation von Server und Agent über Events auf der Basis von WebSockets

Satz:

- Ein Agent spielt gegen einen anderen Agenten über den Server.
- Der Sieger erhält einen Punkt.
- Gibt es keinen eindeutigen Sieger, ermittelt der Server einen Sieger per Zufall

Spiel:

- Das Spiel heißt „4 gewinnt“
- Ein Spiel setzt sich aus drei Sätzen zusammen
- Es gibt einen Hinsatz, den Spieler O beginnt, einen Rücksatz, der von Spieler X begonnen wird sowie einen Zufallssatz, in dem der beginnende Spieler zufällig ausgewählt wird

Spielfeld:

- Das Spielfeld setzt sich aus 7 Spalten und 6 Zeilen zusammen und bietet einen Rahmen, in dem die Spielsteine platziert werden können

Spielstand:

- Angabe der jeweils gewonnenen Punkte im aktuellen Spiel (jeder gewonnene Satz gibt einen Punkt)
- z. B. 1:2, 0:2 etc.

**Thema:**

- Beschreibt das Aussehen und die stilistische Gestaltung der graphischen Oberfläche
- Lässt sich über das Menüband der Hauptseite ändern

Zugzeit:

- Zeit, die der Server nach einem Zug eines Agenten wartet
- Datei-Schnittstelle: Nach dem Ablauf der Zugzeit prüft der Server, ob die Textdatei im spezifischen Pfad beschrieben wurde.
- Push-Schnittstelle: Der Server wartet bis zum Ablauf der Zugzeit und prüft dann, ob er ein Event vom Client empfangen hat.

15. Referenzen

Balzert, H. (2000): Lehrbuch der Software-Technik, Software-Entwicklung.(2. Aufl.), Spektrum Akademischer Verlag.

Lauterbach, M. (2015): Anforderungsbeschreibung Auftrag zum Bau eines Anwendungssystems in Form eines Software-Agenten. (Version V6.00), DHBW Mannheim.



Importanleitung



(WWI14SCA, Praxisprojekt 2016)

Projekt: **VIER.**

Auftraggeber: **DHBW Mannheim**

Auftragnehmer: **WWI14SCA- Team VIER.**

Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub

Version	Datum	Autor	Kommentar
0.1	25.09.2016	A. Neumann	Es handelt sich um die Importanleitung für den Prototyp
0.2	24.10.2016	A. Neumann	Es handelt sich um die Importanleitung für die Beta
0.3	11.11.2016	A. Neumann	Importanleitung für die Releaseversion



1. Einleitung:

Es handelt sich hierbei um eine Anleitung zum rechnerübergreifenden Transfer des Java-Projektes VierPunkt der Gruppe VIER.

2. Systemvoraussetzungen:

- Betriebssystem: ab Microsoft Windows 7
- Browser: Internet Explorer, Firefox Mozilla, Google Chrome
- Entwicklungsumgebung: Eclipse SDK, Version: 4.6.1

3. Projekt auf dem PC speichern

Das Projekt soll aus einer Archivdatei verfügbar gemacht werden. Dazu ist entweder im verwendeten Browser die folgende Adresse einzugeben:

<https://github.com/SvenC56/vierpunkt>

Anschließend rücken Sie bitte „Clone or download“ und wählen dann „Download ZIP“ (s. Abbildung 6).

The screenshot shows a GitHub repository page for 'SvenC56 / vierpunkt'. At the top, there are statistics: 38 commits, 1 branch, 0 releases, and 3 contributors. Below this, there are navigation links for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', and 'Graphs'. A note says 'No description or website provided.' On the right, there are buttons for 'Unwatch', 'Star', 'Fork', and a link to 'Clone with HTTPS'. A prominent green button labeled 'Clone or download' is visible. Below the stats, a list of files includes '.metadata', '.settings', 'Server', and 'bin'. To the right of the file list, there are buttons for 'Open in Desktop' and 'Download ZIP'. A timestamp '6 days ago' is shown at the bottom right.

Abbildung 6: Archivdatei auf den PC runterladen

Die Archivdatei sollte dann mit dem Windows-Explorer entpackt und anschließend an beliebiger Stelle außerhalb des Workspace gespeichert werden.

Wird das Projekt über einen USB-Stick importiert, speichern Sie bitte das Projekt lokal auf Ihrem Rechner.



4. Das Projekt nach Eclipse importieren

Öffnen Sie Eclipse und befolgen Sie folgende Eingaben:

- File → Import → General → Existing Projects into Workspace → Next
- Select root directory wählen → den Speicherort der Archivdatei eingeben
- Häkchen für das zu importierende Projekt **VierPunkt** setzen
- Finish klicken

→ Das importierte Projekt ist nun in Eclipse verfügbar.

5. Applikation starten

Der Agenten wird nun über die Klasse *MainApplication* im Packet *de.dhbw.mannheim.vierpunkt.application* gestartet.



Benutzerdokumentation



(WWI14SCA, Praxisprojekt 2016)

Projekt: **VIER.**

Auftraggeber: **DHBW Mannheim**

Auftragnehmer: **WWI14SCA- Team VIER**

Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub

Version	Datum	Autor	Kommentar
0.1	07.11.2016	A. Neumann	Neu angelegt
0.2	07.11.2016	A. Neumann	Spielkonfiguration zugefügt
0.3	09.11.2016	A. Neumann	Texte angepasst
0.4	10.11.2016	A. Neumann	Finalisiert, Screenshots eingefügt



Inhalt

Inhalt.....	26
Abbildungsverzeichnis	27
1. Allgemeine Problemstellung	28
2. Spielregeln.....	28
3. Ablaufbedingungen.....	29
4. Installation des Programms.....	29
5. Start der Software.....	30
6. Anmeldung der Spieler.....	30
7. Aufbau des Hauptbildschirms.....	31
8. Konfiguration des Spiels.....	34
8.1. Manuelles Spiel	34
8.2. Automatisches Spiel	35
9. Spiel starten.....	37
9.1. Manuelles Spiel	37
9.2. Automatisches Spiel	38
10. Nachverfolgen vergangener Spiele.....	39
11. Verlassen des Spiels.....	40
12. Fehlermeldungen.....	40
12.1. Fehlerhafte Eingabe der Spielernamen.....	40
12.1.1. Eingabe keines bzw. nur eines Spielernamens	40
12.1.2. Eingabe eines identischen Spielernamens.....	41
12.2. Fehlender Kontaktpfad bei Konfiguration der File-Schnittstelle	41
12.3. Auswahl des Spielmodus „Gegen den Computer“.....	41
12.4. Der Pusher-Server reagiert nicht.....	42
12.5. Der Server reagiert nicht	42
13. Mit dem Agenten am Turnier teilnehmen	42



Abbildungsverzeichnis

Abbildung 1: Anwendungsfalldiagramm	10
Abbildung 2: Dialogstruktur der Hauptseite	15
Abbildung 3: Hauptbildschirm.....	16
Abbildung 4: Reiter Optionen.....	16
Abbildung 5: Reiter Themen.....	16
Abbildung 6: Archivdatei auf den PC runterladen.....	23
Abbildung 7: Gewinnsituation "4 gewinnt".....	28
Abbildung 8: Anmeldemaske	30
Abbildung 9: Aufbau des Hauptbildschirms	31
Abbildung 10: Reiter Optionen.....	32
Abbildung 11: Reiter Themen	32
Abbildung 12: Hauptbildschirm in Thema Süßigkeiten.....	32
Abbildung 13: Hauptbildschirm in Thema Food.....	33
Abbildung 14: Hauptbildschirm in Thema Sports.....	33
Abbildung 15: Warnmeldung bei Themenwechsel im laufenden Spiel	34
Abbildung 16: Auswahl des Spielmodus.....	34
Abbildung 17: Einstellungen für ein automatisches Spiel	35
Abbildung 18: Angabe des Kontaktpfads	36
Abbildung 19: Einstellungsoptionen der Pusher-Schnittstelle.....	37
Abbildung 20: Manuelles Spiel starten	38
Abbildung 21: Vergangenes Spiel wieder aufnehmen	39
Abbildung 22: Nachverfolgen vergangener Spiele.....	39
Abbildung 23: Warnmeldung Spiel verlassen.....	40
Abbildung 24: Fehlermeldung Eingabe Spielernamen 1	40
Abbildung 25: Fehlermeldung Eingabe Spielernamen 2	41
Abbildung 26: Fehlermeldung fehlender Kontaktpfad	41
Abbildung 27: Fehlermeldung Spielmodus	41
Abbildung 28: Fehlermeldung Pusher-Server	42
Abbildung 29: Sequenzdiagramm Satzbeginn.....	43
Abbildung 30: Sequenzdiagramm Spielzug	44
Abbildung 31: Sequenzdiagramm Satzende.....	45
Abbildung 32: Fachlicher Kontext	53
Abbildung 33: High System Level Overview	54
Abbildung 34: Produktstruktur VIER.	56
Abbildung 35: Fachklassendiagramm VIER.	57
Abbildung 36: Ausschnitt eines Spielbaums von Tic Tac Toe	60
Abbildung 37: Beispiel für einen Alpha-Beta-Cut.....	61
Abbildung 38: Beispiel Berechnung isRow().....	64
Abbildung 39: Aufbau der GUI	73
Abbildung 40: Auszug Coding ChangeListener	74
Abbildung 41: Auszug Coding EventHandler	74
Abbildung 42: Auszug Coding Login	75
Abbildung 43: Auszug Coding GridPane	76
Abbildung 44: Auszug Coding Methoden auf GridPane.....	77
Abbildung 45: Aufbau Menü	78
Abbildung 46: ER-Diagramm der Entitäten	81
Abbildung 47: Aktivitätendiagramm "Satz spielen"	83
Abbildung 48: Aktivitätendiagramm "Spiel nachverfolgen"	84



1. Allgemeine Problemstellung

Das vorliegende Programm bietet die Möglichkeit das Spiel „4 gewinnt“ autonom gegen einen anderen Spieler zu spielen. Die Kommunikation beider Spieler findet dabei über einen zwischengeschalteten Server statt, mit dem beide Spieler verbunden sind.

Die Spielzüge des Agenten (also der spielenden künstlichen Intelligenz, KI) sowie des Gegners können über ein graphisches Spielfeld nachverfolgt werden. Über eine automatische Speicherung der Spielzüge sowie Punkte in einer Datenbank können bereits gespielte Spiele im Anschluss nochmals nachverfolgt werden.

Der so programmierte Softwareagent soll anschließend an einem Turnier gegen die Agenten anderer Gruppen antreten (vgl. Kapitel 13).

2. Spielregeln

Das Spiel VIER. orientiert sich an den Spielregeln einer Partie „4 gewinnt“.

Ein Spieler spielt ein Spiel gegen einen Gegner. Jedes Spiel besteht dabei aus drei Sätzen, die bei Gewinn mit je einem Punkt belohnt werden. Wer zuerst zwei der drei gespielten Sätze

gewonnen hat, gewinnt das Spiel.

Um einen Satz für sich zu entscheiden, muss der Spieler vor dem Gegner auf dem Spielfeld vier Spielsteine nebeneinander (horizontal, vertikal oder diagonal) platzieren (vgl. Abbildung 7). Ist das Spielfeld voll ohne dass einer der beiden Spieler vier Steine in Reihe platzieren konnte, wird der Gewinner des Satzes per Zufall ermittelt.

*Abbildung 7: Gewinnsituation
"4 gewinnt"*



3. Ablaufbedingungen

Für einen fehlerfreien Ablauf des Programms werden folgende Hard- und Softwarekomponenten vorausgesetzt:

Hardware

- PC
- Intel Xeon Prozessor E3-1245, 64 bit
- CD-ROM-Laufwerk zur Installation
- Maus & Tastatur
- Monitor

Software

- Betriebssystem:
 - Windows 7 oder MAC OS X
- Laufzeitumgebung ab Java SE 7 (dementsprechend mit laufendem JavaFx-Framework)
- Datenbank HSQLDB (Version 2.3.4)
- Eclipse Neon

4. Installation des Programms

Um mit dem Agenten das Spiel „4 gewinnt“ spielen zu können, ist der Agent zunächst zu installieren. Zur Installation des Programms muss das Verzeichnis „Vierpunkt-Master“ inklusive der darin enthaltenden Dateien und Ordner von der Installations-CD auf die Festplatte kopiert werden.

Anschließend ist das Projekt in Eclipse zu importieren. Dazu muss Eclipse geöffnet werden.

Über den Reiter „File“ der Menüleiste ist anschließend „Import“ auszuwählen.

Aus der Liste aller angegebenen Möglichkeiten ist „General“ zu wählen und anschließend „Existing Projects into Workspace“. Über „Next“ geht es weiter.

Anschließend ist „Select root directory“ auszuwählen und der Speicherort des Verzeichnis „Vierpunkt-Master“ auszuwählen. Alle möglichen zu importierenden Projekte werden anschließend in der unteren Auflistung angezeigt.

Das Häkchen ist bei VierPunkt zu setzen und der Import durch das Betätigen von „Finish“ abzuschließen.



5. Start der Software

Ist das Projekt in Eclipse importiert, lässt sich die Applikation über die Klasse *MainApplication.java*, die in Paket *de.dhbw.vierpunkt.application* abgelegt ist, starten. Dazu wählen Sie bitte die Datei in Eclipse aus und starten Sie diese über „Run as JavaApplication“.

6. Anmeldung der Spieler

Nach Starten der Klasse hat der Nutzer über einen Anmeldebildschirm die Möglichkeit sowohl seinen eigenen Namen als auch den des Gegners anzugeben (s. Abbildung 8).



Abbildung 8: Anmeldemaske

Wurden entweder nicht beide Felder ausgefüllt oder wurde der gleiche Name an beide Teilnehmer vergeben, werden entsprechende Warnmeldungen ausgegeben (s. Kapitel 12.1). Um fortfahren zu können, sind zwei unterschiedliche Spielernamen einzutragen.

Nach Eintrag beider Spielernamen können Sie entweder über „Spiel starten“ oder die Returntaste Ihrer Tastatur fortfahren.



7. Aufbau des Hauptbildschirms



Abbildung 9: Aufbau des Hauptbildschirms

Nach erfolgreicher Anmeldung der Spieler öffnet sich automatisch der Hauptbildschirm des Spiels VIER. Auf diesem sind neben dem Spielfeld noch weitere Optionen zur Einstellung abgebildet (s. Abbildung 9).

Auf der linken Seite des Spielfelds wird angezeigt, welche Spieler gegeneinander antreten und wem dabei welche Spielsteine zugeordnet sind.

Rechts neben dem Spielfeld können Sie den aktuellen Punktestand sowie den Satzstatus des Spiels einsehen. Der Satzstatus wechselt dabei im Laufe eines Spiels zwischen „Satz spielen“ und „Satz beendet“. Wurde noch kein Spiel begonnen, gibt er „Spiel noch nicht begonnen“ an.

Darunter haben Sie zusätzlich die Möglichkeit zu bestimmen, in welchem Spielmodus (manuell oder automatisch) Sie VIER gerne spielen würden.

Eine Menüleiste am oberen Rand des Hauptbildschirms erlaubt Ihnen weitere Funktionen.

Unter dem Reiter „Optionen“ können Sie entweder ein neues Spiel starten, das Spiel verlassen oder ein vergangenes Spiel nachverfolgen (s. Abbildung 10).



Wählen Sie hingegen den Reiter „Themen“, können Sie zwischen vier verschiedenen Designs des Hauptbildschirms wählen (s. Abbildung 11).

Optionen	Themen
neues Spiel	
bereits gespielte Spiele	
Spiel beenden	

Abbildung 10: Reiter Optionen

VierPunkt	Themen
	Suessigkeiten
	Halloween
	Food
	Sports

Abbildung 11: Reiter Themen

Wählen Sie eines der Themen aus, die im Reiter „Themen“ des Menübands zur Auswahl stehen, ändert sich die Oberfläche des Hauptbildschirms (s. Abbildung 12-14).



Abbildung 12: Hauptbildschirm in Thema Süßigkeiten

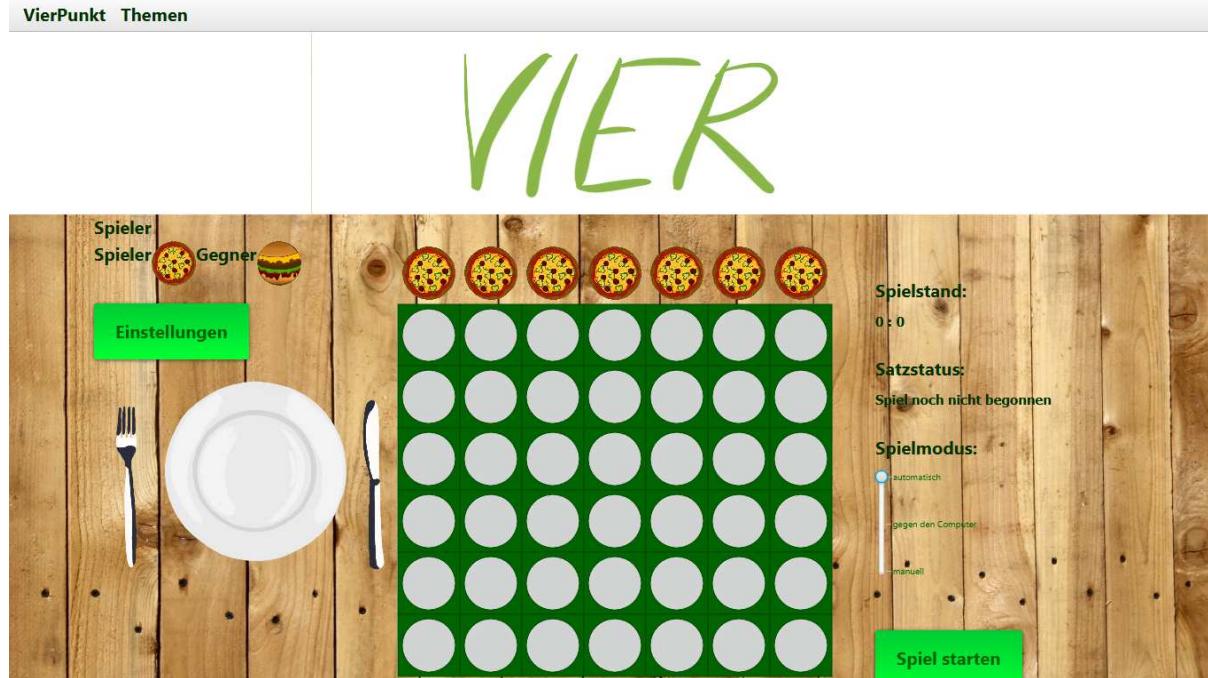


Abbildung 13: Hauptbildschirm in Thema Food

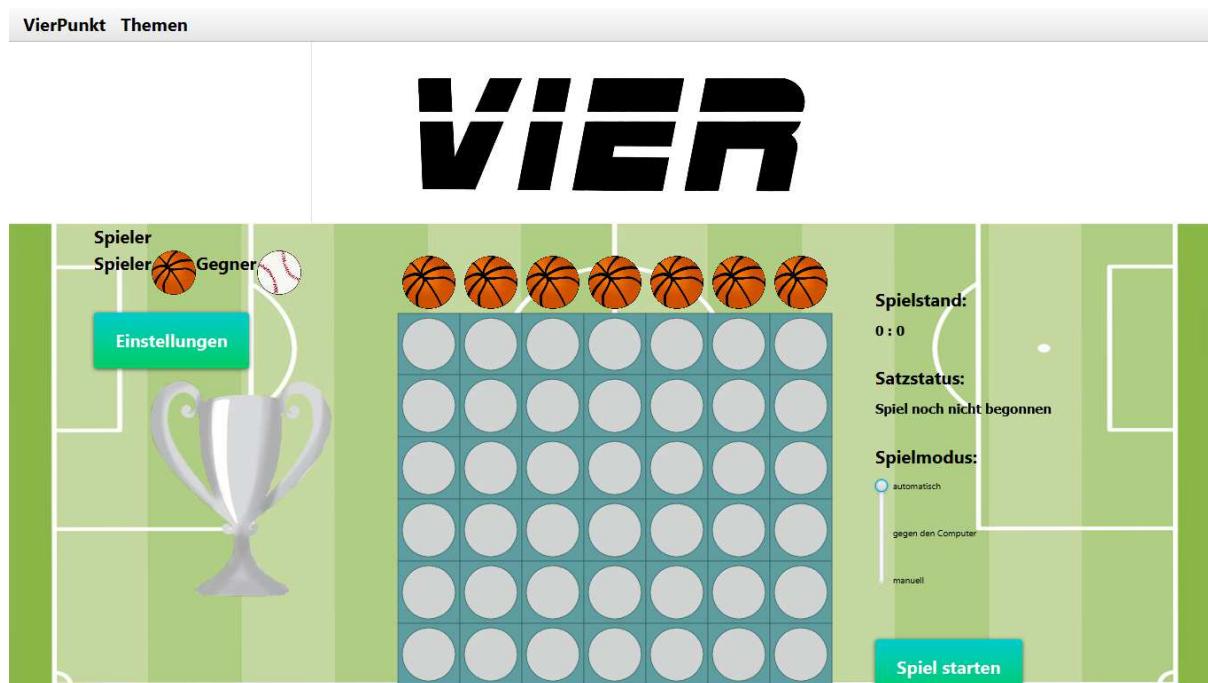


Abbildung 14: Hauptbildschirm in Thema Sports

Bitte beachten Sie, dass ein Wechsel des Themas während eines laufenden Spiels dazu führt, dass das Spiel vorzeitig abbricht. Das Programm weist Sie darauf über eine Warnmeldung hin (s. Abbildung 15).

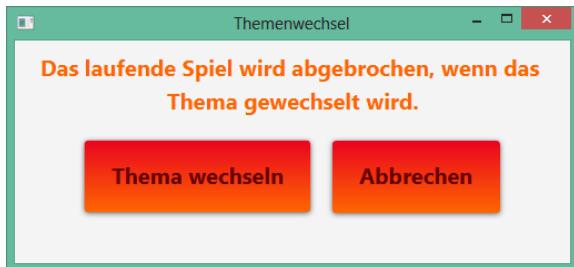


Abbildung 15: Warnmeldung bei Themenwechsel im laufenden Spiel

Der Hauptbildschirm startet im Vollbildmodus. Möchten Sie diesen verlassen, drücken Sie bitte die Escape-Taste auf Ihrer Tastatur.

8. Konfiguration des Spiels

Die Applikation VIER. ermöglicht Ihnen das Spiel in zwei verschiedenen Modi. Die Konfiguration, die für ein erfolgreiches Spiel durchzuführen ist, passt sich dabei an den von Ihnen gewählten Modus an.



Der Modus lässt sich über die Leiste auf der rechten Seite des Spielfelds auswählen (s. Abbildung 16).

Der Modus „Gegen den Computer“ ist allerdings in dieser Version noch nicht implementiert und wird erst in Version 2.0 von VIER. verfügbar sein. Wählen Sie dennoch diesen Modus an, generiert VIER. eine Fehlermeldung (s. Kapitel 12.3).

Abbildung 16: Auswahl des Spielmodus

8.1. Manuelles Spiel

Entscheiden Sie sich für ein manuelles Spiel, können Sie auf dem gleichen Rechner gegen einen Gegner spielen. Die Spielsteine werden hierbei durch Klicken auf die von Ihnen bevorzugte Position im Spielfeld gesetzt. Um ein manuelles Spiel zu starten, wählen Sie bitte den Modus „manuell“.



8.2. Automatisches Spiel

Wählen Sie ein automatisches Spiel, spielt die künstliche Intelligenz in Ihrem Namen gegen die künstliche Intelligenz eines anderen Spielers an einem anderen Rechner.

Sie haben nun die Möglichkeit weitere Einstellungen vorzunehmen. Dazu wählen Sie bitte den Knopf „Einstellungen“ auf der linken Seite des Spielfelds (s. Abbildung 9).

Es öffnet sich nun ein weiteres Fenster, in dem Sie Einstellungen vornehmen können. Geben Sie bitte an, ob der Agent als Spieler X oder Spieler O agiert, da so vom Server ermittelt werden kann, welcher Spieler mit dem ersten Zug beginnt (s. Abbildung 17).

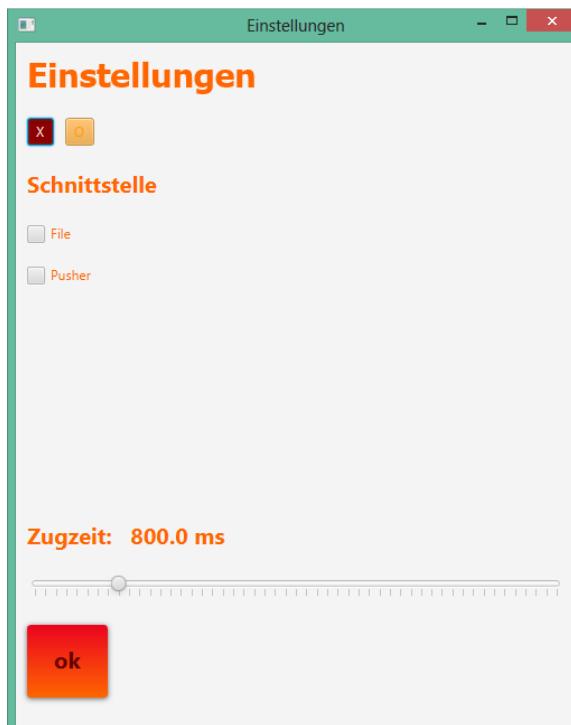


Abbildung 17: Einstellungen für ein automatisches Spiel

Um die Kommunikation zum Server und damit ein Spiel gegen die gegnerische KI zu ermöglichen, wählen Sie bitte aus, über welche Schnittstelle der Agent mit dem Server kommunizieren soll. Entscheiden Sie sich für die File-Schnittstelle, haben Sie die Möglichkeit in einem weiteren Fenster einen Kontaktpfad anzugeben (s. Abbildung 18).

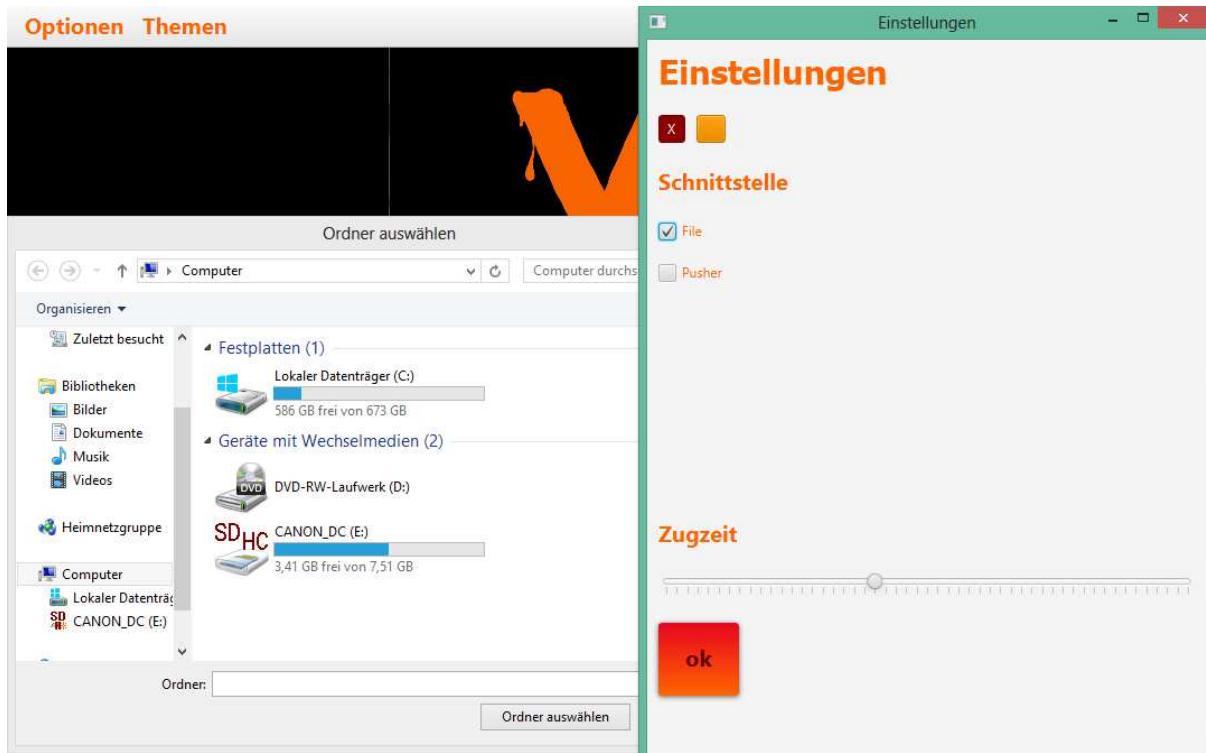


Abbildung 18: Angabe des Kontaktpfads

Sollten Sie keinen Kontaktpfad angegeben aber dennoch die File-Schnittstelle ausgewählt haben, erhalten Sie eine Fehlermeldung durch das System (s. Kapitel 12.2).

Wählen Sie hingegen die Pusher-Schnittstelle, erscheinen weitere Eingabefelder, in welchen Sie eine App ID, den App Key und das App Secret eintragen können. Standardmäßig sind hier Defaultwerte hinterlegt, die Sie allerdings sofern gewünscht überschreiben können (s. Abbildung 19).



The screenshot shows a configuration window for a Pusher service. At the top left is a 'File' menu item. Next to it is a checked checkbox labeled 'Pusher'. Below these are four input fields: 'App ID:' containing '255967', 'App Key:' containing '61783ef3dd40e1b399b2', and 'App Secret:' containing '66b722950915220b298c'. The entire window has a light gray background with dark gray borders around the input fields.

Abbildung 19: Einstellungsoptionen der Pusher-Schnittstelle

Haben Sie sich für eine Schnittstelle entschieden, ist zusätzlich über die Zeitleiste die Zugzeit des Agenten einzustellen (s. Abbildung 17). Ist die Zugzeit ausgewählt, wird diese als Zahlenwert angezeigt.

Entsprechen alle Einstellungen Ihren Vorstellungen, bestätigen Sie bitte mit „Ok“, um das Einstellungsfenster zu schließen und zurück zum Hauptbildschirm zu gelangen.

9. Spiel starten

In Abhängigkeit von dem Modus, in dem Sie spielen wollen, wird das Spiel auf unterschiedliche Weise gestartet.

9.1. Manuelles Spiel

Möchten Sie ein manuelles Spiel beginnen, klicken Sie bitte entweder mit Ihrem Mauszeiger auf die Zelle im Spielfeld, an der Sie Ihren Spielstein platzieren wollen oder auf die Steinvororschau, die oberhalb des Spielfelds abgebildet wird (s. Abbildung 20).



Abbildung 20: Manuelles Spiel starten

Es ist zu beachten, dass Spieler 1 (der Spieler, dem bei Anmeldung der Namen des „Spieler“ zugeordnet wurde) stets anfängt. Der Gegner kann anschließend durch einen weiteren Klick seinen Stein platzieren. Die Spieler alternieren so lange, bis entweder ein Spieler vier Steine in einer Reihe setzen konnte oder das Spielfeld komplett mit Steinen gefüllt ist.

9.2. Automatisches Spiel

Wurde das Spiel wie in Kapitel 8.2 beschrieben entsprechend konfiguriert, kann ein Spiel gegen einen Gegner durch „Spiel starten“ auf dem Hauptbildschirm gestartet werden. Die KI wird nun die Spielsteine automatisch auf dem Spielfeld setzen und das Spiel eigenständig zu Ende spielen.

Wurde allerdings ein vorheriges Spiel nicht beendet, werden Sie durch eine Systemnachricht nach Betätigen von „Spiel starten“ darüber in Kenntnis gesetzt und können dieses erneut aufnehmen (s. Abbildung 21). Wollen Sie dies nicht, drücken Sie bitte „Nein, Spiel verwerfen“. Das Fenster wird sich dann schließen und Sie können ein neues Spiel mit „Spiel starten“ starten.



Abbildung 21: Vergangenes Spiel wieder aufnehmen

10. Nachverfolgen vergangener Spiele

Soll ein bereits gespieltes Spiel nachvollzogen werden, wählen Sie bitte im Reiter Optionen der Menüleiste die Option „Vergangene Spiele nachverfolgen“ (s. Abbildung 10).

Es öffnet sich nun ein neues Fenster, in welchem Ihnen die letzten zehn gespielten Spiele angezeigt werden. Möchten Sie ein spezifisches Spiel nachverfolgen, wählen Sie es bitte in der Liste durch einen Klick an.

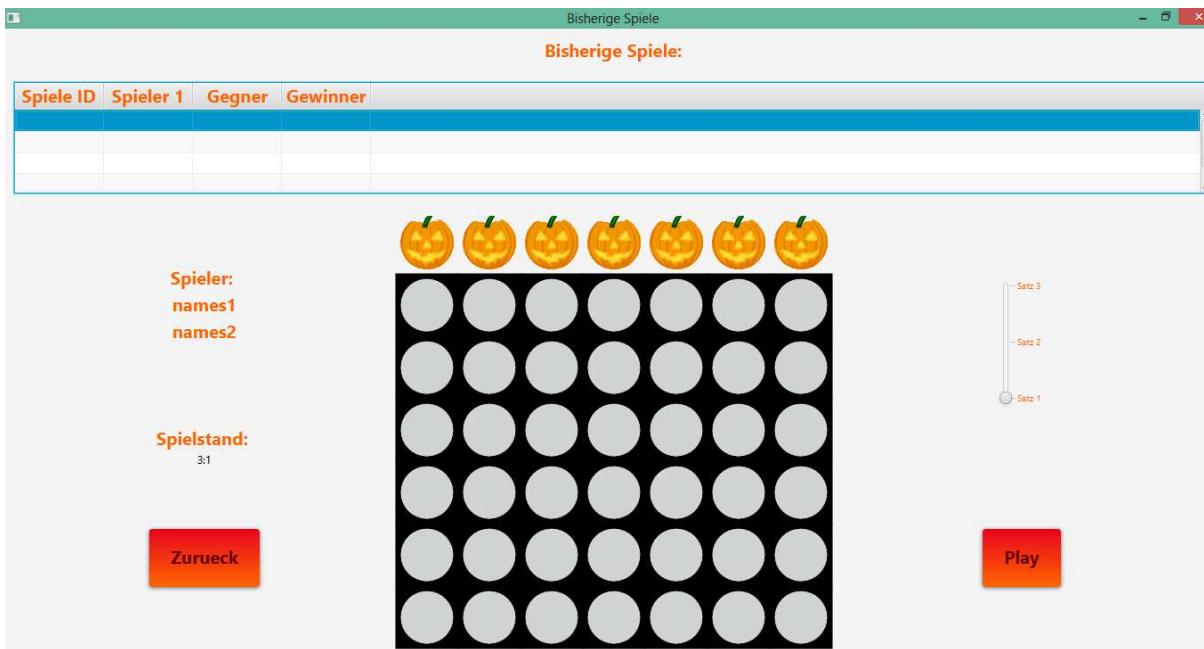


Abbildung 22: Nachverfolgen vergangener Spiele

Die Leiste, die sich rechts vom Spielfeld befindet, ermöglicht es Ihnen den Satz auszuwählen, den Sie nachverfolgen wollen. Drücken Sie anschließend auf den Knopf „Play“, um die automatische Wiedergabe zu starten. Wenn Sie zum Hauptbildschirm zurückkehren wollen, können Sie dies über den Button „Zurück“ erreichen (s. Abbildung 22).



11. Verlassen des Spiels

Das Spiel kann über den Reiter „Optionen“ und anschließend „Spiel verlassen“ beendet werden. Um zu verhindern, dass Spiele, die noch laufen, nicht unbeabsichtigt abgebrochen werden, fragt VIER. ob Sie das Spiel wirklich beenden wollen (s. Abbildung 23). Trifft dies zu, bestätigen Sie bitte „Beenden“.



Abbildung 23: Warnmeldung Spiel verlassen

12. Fehlermeldungen

In VIER. können folgende Fehlersituationen auftreten.

12.1. Fehlerhafte Eingabe der Spielernamen

12.1.1. Eingabe keines bzw. nur eines Spielernamens

- **Fehlermeldung:**
Bitte Spielernamen eingeben
- **Fehlerursache:**
Es wurde bei der Anmeldung kein oder nur ein Spielername eingetragen.
- **Behebungsmaßnahme:**
Geben Sie sowohl sich als auch Ihrem Gegner einen Namen.



Abbildung 24: Fehlermeldung Eingabe Spielernamen 1



12.1.2. Eingabe eines identischen Spielernamens

- **Fehlermeldung:**

Bitte unterschiedliche Spielernamen wählen.

- **Fehlerursache:**

Es wurde der identische Name für beide Spieler vergeben.

- **Behebungsmaßnahme:**

Sorgen Sie dafür, dass sich die beiden Spie- lernamen voneinander unterscheiden.



Abbildung 25: Fehlermeldung Eingabe Spielernamen 2

12.2. Fehlender Kontaktpfad bei Konfiguration der File-Schnittstelle

- **Fehlermeldung:**

Achtung! Es wurde kein Ordner ausgewählt!



Abbildung 26: Fehlermeldung fehlender Kontaktpfad

- **Fehlerursache:**

Es wurde bei Konfiguration der File- Schnittstelle kein Kontaktpfad angege- ben.

- **Behebungsmaßnahme:**

Wählen Sie einen Kontaktpfad aus, über den der Agent mit dem Server kommunizieren soll.

12.3. Auswahl des Spielmodus „Gegen den Computer“

- **Fehlermeldung:**

Diese Funktion wurde noch nicht implemen- tiert. Bitte wähle einen anderen Spielmodus.

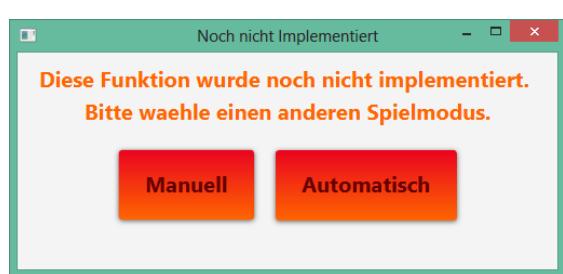


Abbildung 27: Fehlermeldung Spielmodus

- **Fehlerursache:**

Es wurde der Spielmodus „Gegen den Com- puter“ ausgewählt.

- **Behebungsmaßnahme:**

Dieser Modus ist in Version 1 der Applikation



VIER. noch nicht implementiert. Wählen Sie bitte einen anderen der beiden Spielmodi.

12.4. Der Pusher-Server reagiert nicht

- **Fehlermeldung:**

Der Pusher-Server reagiert nicht, bitte Pusher Credentials in den Einstellungen überprüfen.



Abbildung 28: Fehlermeldung Pusher-Server

- **Fehlerursache:**

Der Pusher-Server antwortet nicht.

- **Behebungsmaßnahme:**

Überprüfen und ändern Sie bitte die Pusher Credentials unter „Einstellungen“.

12.5. Der Server reagiert nicht

- **Fehlermeldung:**

Server vermutlich abgestürzt. Bitte Programm erneut starten.

- **Fehlerursache:**

Es gibt ein Time-Out, da die Antwort des Servers zu lange dauert.

- **Behebungsmaßnahme:**

Starten Sie das Spiel erneut.

13. Mit dem Agenten im Turnier spielen

Der Agent spielt im Rahmen eines Turniers gegen die Agenten der anderen Gruppen.

Um am Turnier teilzunehmen, ist der Agent wie beschrieben über den Server anzubinden. Je nachdem, ob der Agent als Spieler X oder Spieler O agiert, setzt er entweder den ersten Stein oder wartet auf den ersten Zug des Gegners.

Der Ablauf des Spielens im Turnier ist in den folgenden Sequenzdiagrammen für die Situationen Satzbeginn, Spielzug spielen sowie Satzende dargestellt (s. Abbildung 29-31). Es wird zwischen den drei Objekten Agent, Server sowie Dozent unterschieden. Die Sequenzdiagramme geben einen Überblick über die zeitliche Abfolge der Aktionen der beteiligten Akteure.

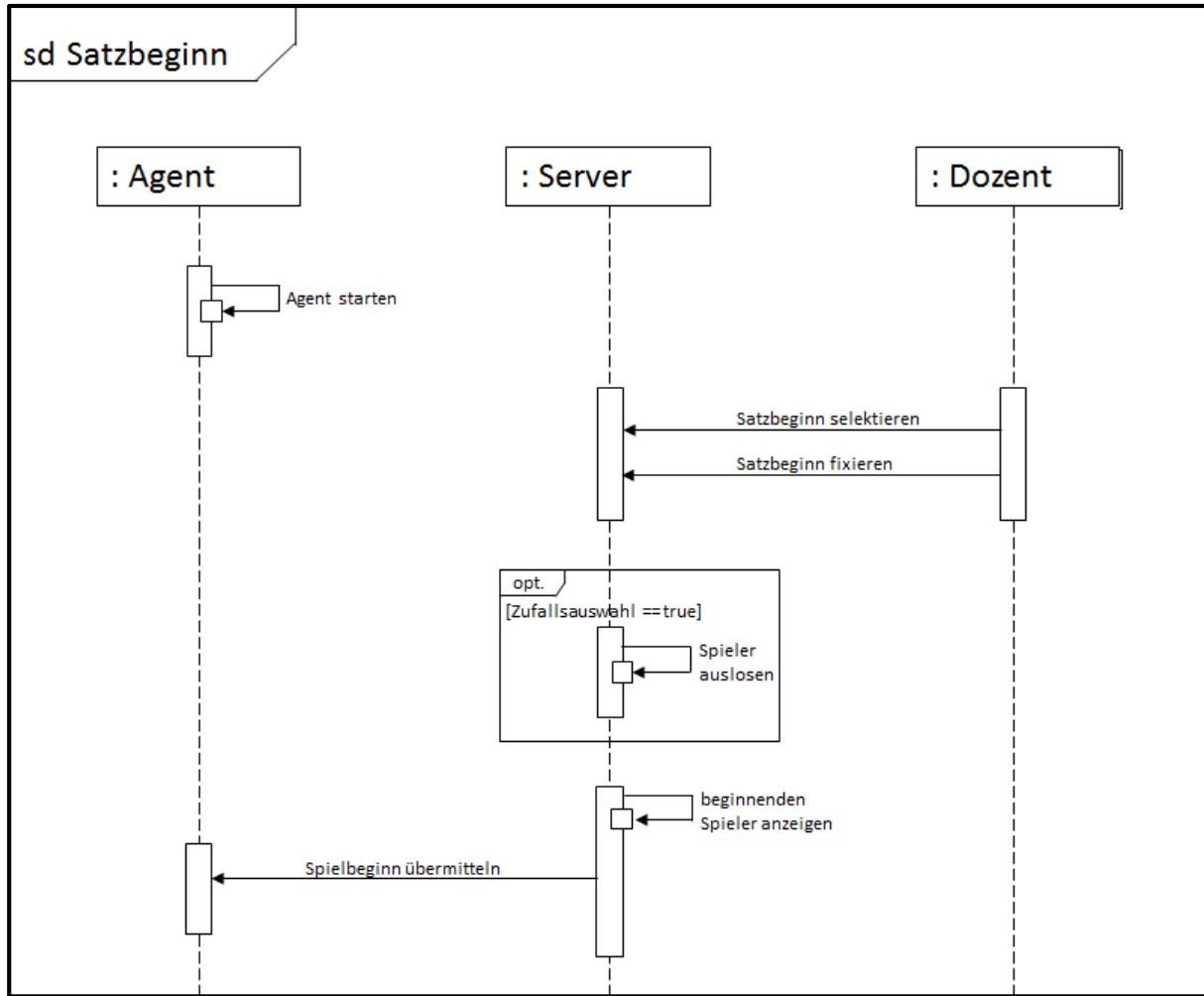


Abbildung 29: Sequenzdiagramm Satzbeginn

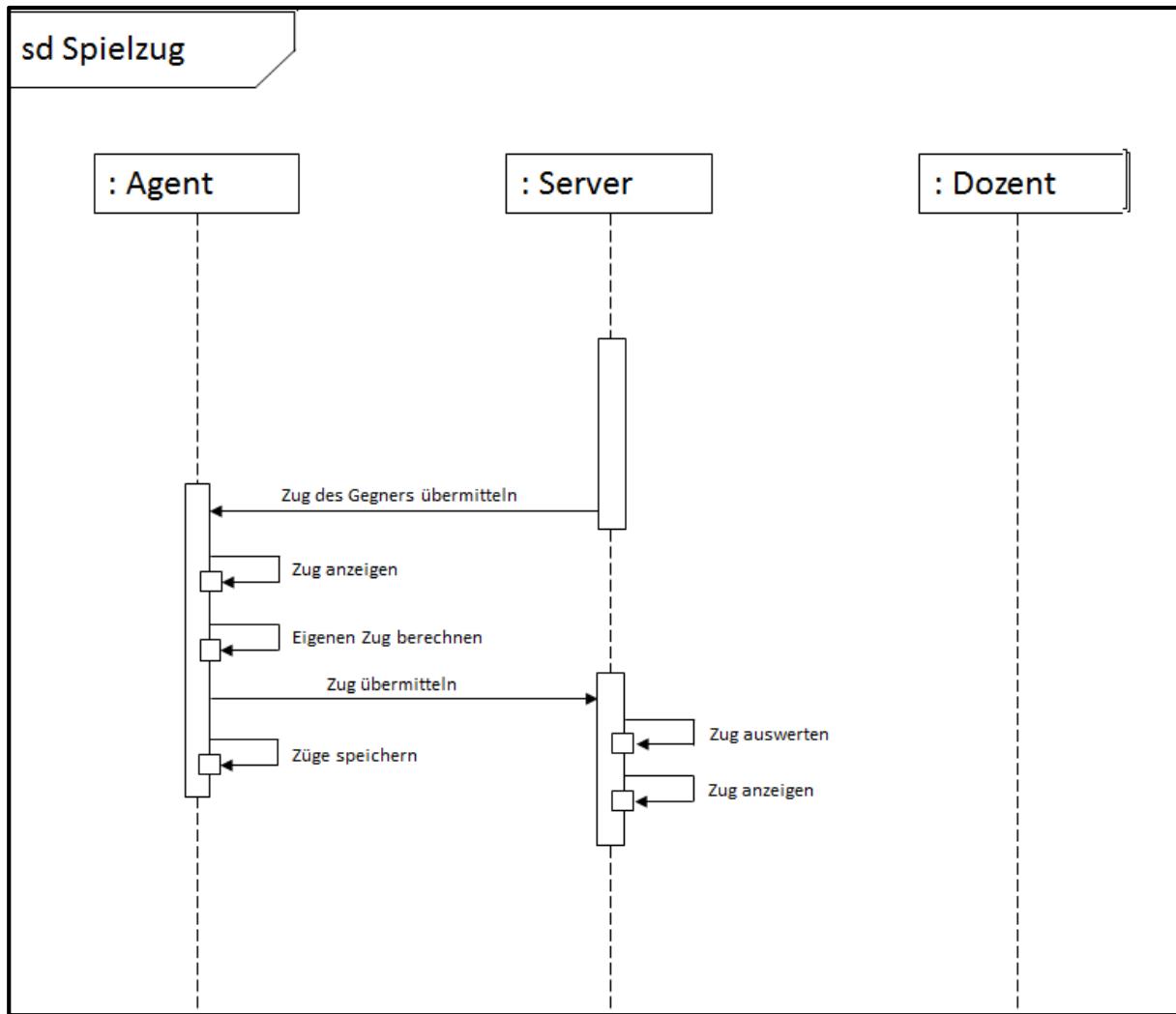


Abbildung 30: Sequenzdiagramm Spielzug

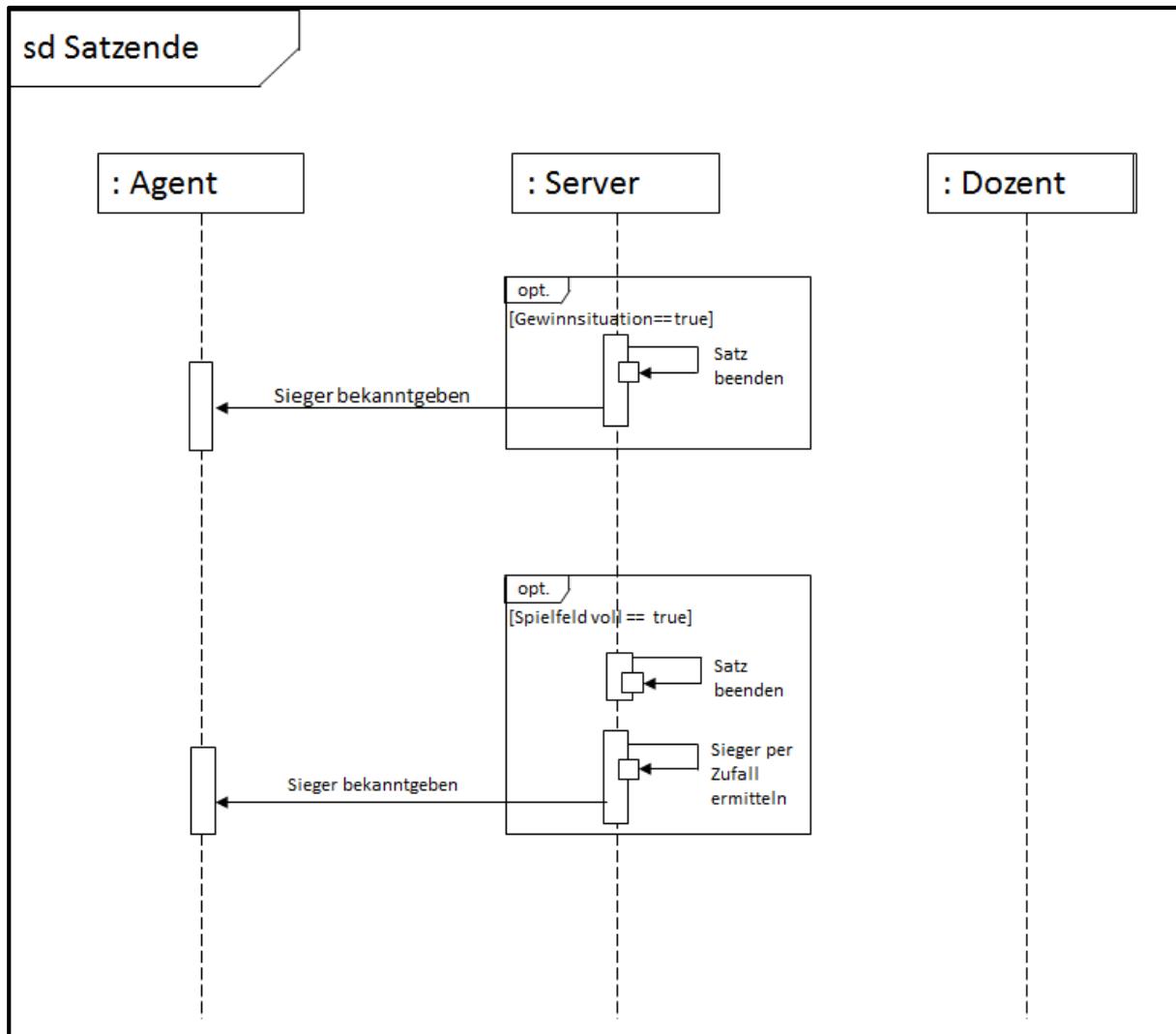


Abbildung 31: Sequenzdiagramm Satzende



Entwicklungsdocumentation

VIER



(WWI14SCA, Praxisprojekt 2016)

Projekt: **VIER.**

Auftraggeber: **DHBW Mannheim**

Auftragnehmer: **WWI14SCA- Team VIER**

Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub

Version	Datum	Autor	Kommentar
0.1	07.11.2016	A. Neumann	Neu angelegt
0.2	07.11.2016	A. Neumann	High System Level Overview ergänzt
0.3	09.11.2016	A. Neumann	Texte angepasst
0.4	10.11.2016	A. Neumann	Graphiken eingefügt
0.5	12.11.2016	S. Cieslok, T. Jung, J. Schaub, L. Birk, J. Lang	Klassenbeschreibung ergänzt



Inhalt

Inhalt.....	47
Abbildungsverzeichnis	49
1. Einführung	50
1.1. Was ist VIER.?	50
1.2. Wesentliche Features.....	50
1.3. Qualitätsziele	50
1.4. Stakeholder	51
2. Randbedingungen.....	51
2.1. Technische Randbedingungen	51
2.2. Organisatorische Randbedingungen.....	52
2.3. Konventionen	52
3. Fachlicher Kontext	53
3.1. Menschlicher Gegner (Benutzer)	53
3.2. KI Gegner (Fremdsystem).....	53
4. High System Level Overview.....	54
4.1. Logik.....	54
4.2. Schnittstellen.....	54
4.3. Graphische Oberfläche.....	55
4.4. Datenhaltung.....	55
5. Produktstruktur	55
6. Fachklassen	56
7. Konzeption	58
7.1. Die Startklasse	58
7.2. Die Spiellogik	59
7.2.1. Der Algorithmus – Klasse „AlphaBeta“.....	59
7.2.2. Das Spielobjekt – Klasse „Game“.....	62
7.2.3. Das Satzobjekt – Klasse „Match“	63
7.2.4. Die Schnittstelle – Klasse „NameListener“	65
7.2.5. Das Spielerobjekt – Klasse „Player“.....	65
7.2.6. Eine Spielposition – Klasse „PlaySlot“	66
7.2.7. Der Spielzug – Klasse „Turn“	66
7.3. Schnittstellen.....	67
7.3.1. Klasse PusherInterface.java	68
7.3.2. Klasse FileInterface.java	69
7.3.3. -Interfaces im Package Interfaces	70
7.4. Gui	70



7.4.1.	Stages.....	71
7.4.2.	Layout	72
7.4.3.	EventHandler	73
7.4.4.	Das Spielfeld	75
7.4.5.	Das Menü.....	77
7.5.	Datenbank	78
7.5.1.	ConnectHSQl.java.....	79
7.5.2.	DBConnector.java	80
7.5.3.	Getter – Methoden .java	80
7.5.4.	Setter – Methoden .java	81
7.5.5.	LegacyMethods.java	81
7.5.6.	Datenbank Aufbau	81
8.	Programmablauf	82
8.1.	Satz spielen.....	82
8.2.	Gespielte Spiele nachverfolgen.....	84
9.	JavaDoc	85
10.	Referenz	83



Abbildungsverzeichnis

Abbildung 1: Anwendungsfalldiagramm	10
Abbildung 2: Dialogstruktur der Hauptseite	15
Abbildung 3: Hauptbildschirm.....	16
Abbildung 4: Reiter Optionen.....	16
Abbildung 5: Reiter Themen.....	16
Abbildung 6: Archivdatei auf den PC runterladen.....	23
Abbildung 7: Gewinnsituation "4 gewinnt".....	28
Abbildung 8: Anmeldemaske	30
Abbildung 9: Aufbau des Hauptbildschirms	31
Abbildung 10: Reiter Optionen.....	32
Abbildung 11: Reiter Themen	32
Abbildung 12: Hauptbildschirm in Thema Süßigkeiten.....	32
Abbildung 13: Hauptbildschirm in Thema Food.....	33
Abbildung 14: Hauptbildschirm in Thema Sports.....	33
Abbildung 15: Warnmeldung bei Themenwechsel im laufenden Spiel	34
Abbildung 16: Auswahl des Spielmodus.....	34
Abbildung 17: Einstellungen für ein automatisches Spiel	35
Abbildung 18: Angabe des Kontaktpfads	36
Abbildung 19: Einstellungsoptionen der Pusher-Schnittstelle.....	37
Abbildung 20: Manuelles Spiel starten	38
Abbildung 21: Vergangenes Spiel wieder aufnehmen	39
Abbildung 22: Nachverfolgen vergangener Spiele.....	39
Abbildung 23: Warnmeldung Spiel verlassen.....	40
Abbildung 24: Fehlermeldung Eingabe Spielernamen 1	40
Abbildung 25: Fehlermeldung Eingabe Spielernamen 2	41
Abbildung 26: Fehlermeldung fehlender Kontaktpfad	41
Abbildung 27: Fehlermeldung Spielmodus	41
Abbildung 28: Fehlermeldung Pusher-Server	42
Abbildung 29: Sequenzdiagramm Satzbeginn.....	43
Abbildung 30: Sequenzdiagramm Spielzug	44
Abbildung 31: Sequenzdiagramm Satzende.....	45
Abbildung 32: Fachlicher Kontext	53
Abbildung 33: High System Level Overview	54
Abbildung 34: Produktstruktur VIER.	56
Abbildung 35: Fachklassendiagramm VIER.	57
Abbildung 36: Ausschnitt eines Spielbaums von Tic Tac Toe	60
Abbildung 37: Beispiel für einen Alpha-Beta-Cut.....	61
Abbildung 38: Beispiel Berechnung isRow().....	64
Abbildung 39: Aufbau der GUI	73
Abbildung 40: Auszug Coding ChangeListener	74
Abbildung 41: Auszug Coding EventHandler	74
Abbildung 42: Auszug Coding Login	75
Abbildung 43: Auszug Coding GridPane	76
Abbildung 44: Auszug Coding Methoden auf GridPane.....	77
Abbildung 45: Aufbau Menü	78
Abbildung 46: ER-Diagramm der Entitäten	81
Abbildung 47: Aktivitätendiagramm "Satz spielen"	83
Abbildung 48: Aktivitätendiagramm "Spiel nachverfolgen"	84



1. Einführung

1.1. Was ist VIER.?

Bei VIER. handelt es sich um eine voll funktionsfähige „4 gewinnt“ Engine, bei welcher eine integrierte künstliche Intelligenz (KI) autonom ohne Spielereingriff gegen einen anderen Spieler spielen kann. Neben diesem automatischen Spiel ist zusätzlich auch ein manuelles Spiel am PC gegen einen anderen User möglich.

1.2. Wesentliche Features

- Vollständige Implementierung der Regeln von „4 gewinnt“
- Unterstützt das Spiel gegen andere „4 gewinnt“ Engines
- Erlaubt ein autonomes Spiel durch die KI ohne zusätzlichen Usereingriff
- modernes grafisches „4 gewinnt“-Frontend

1.3. Qualitätsziele

Die folgende Tabelle beschreibt die zentralen Qualitätsziele von VIER. Die Reihenfolge gibt dabei eine grobe Orientierung bezüglich ihrer Wichtigkeit.

Tabelle 2: Qualitätsziele

Qualitätsmerkmal	Motivation und Erläuterung
Zugängliches Beispiel (Analysebarkeit)	Da VIER. in erster Linie als Anschauungsmaterial für Studenten und Dozenten dient, erschließen sich Entwurf und Implementierung rasch.
Einladende Experimentierplattform (Änderbarkeit)	Alternative Algorithmen und Strategien, etwa zur Bewertung einer Steinstellung, können leicht implementiert und in die Lösung integriert werden.
Akzeptable Spielstärke (Attraktivität)	VIER. spielt stark genug, um schwache Gegner sicher zu schlagen und Gelegenheitsspieler zumindest zu fordern.
Schnelles Antworten auf Züge (Effizienz)	Da VIER. gegen den Timeout durch den Servers arbeitet, erfolgt die Berechnung der Spielzüge entsprechend schnell.



1.4. Stakeholder

Die folgende Tabelle gibt einen Überblick über die Stakeholder von VIER. sowie deren jeweilige Intention.

Tabelle 3: Stakeholder

Wer?	Interesse und Bezug
Gruppe VIER.	<ul style="list-style-type: none">• Bearbeitung der Aufgabenstellung• Einblicke in die Softwareentwicklung• Teilnahme am Turnier
Dozent der DHBW	<ul style="list-style-type: none">• Bewertung des Erfolgs der Applikation• Veranstaltung eines Turniers

2. Randbedingungen

Beim Lösungsentwurf sind verschiedene Randbedingungen zu beachten, welche im Folgenden erläutert werden.

2.1. Technische Randbedingungen

Die Technischen Randbedingungen geben einen Überblick über den Betrieb im System sowie die Entwicklungsbedingungen.

Tabelle 4: Technische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Moderate Hardwareausstattung	Betrieb der Lösung auf einem marktüblichen PC im Computerlabor der DHBW.
Betrieb auf Windows-Desktop-Betriebssystemen	Standardausstattung der PCs im Computerlabor. Unterstützung anderer Betriebssysteme (allen voran Linux und Mac OS X) wünschenswert, aber nicht zwingend erforderlich.
Implementierung in Java	Funktionale Anforderung der Applikation. Entwicklung unter Version Java SE 8 (betrifft VIER. Version 1.0).



2.2. Organisatorische Randbedingungen

Die organisatorischen Randbedingungen geben einen Überblick über die Gestaltung des Projekts.

Tabelle 5: Organisatorische Randbedingungen

Randbedingung	Erläuterungen, Hintergrund
Team	Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang, Angelina Neumann, Jana Schaub
Zeitplan	Beginn der Entwicklung September 2016, erster lauffähiger Prototyp Oktober 2016 Fertigstellung Version 1.0: 14. November 2016 (Abgabe der gesamten Dokumentation)
Vorgehensmodell	Entwicklung nach dem Wasserfallmodell Die Dokumentation orientiert sich in Ansätzen an arc42 .
Entwicklungswerzeuge	Entwurf mit Stift und Papier Erstellung der Java-Quelltexte in Eclipse Integration der HSQLDB
Konfigurations- und Versionsverwaltung	Git bei GitHub

2.3. Konventionen

Folgenden Konventionen wurden bei der Implementierung von VIER. gefolgt:

Tabelle 6: Konventionen

Konvention	Erläuterungen, Hintergrund
Kodierrichtlinien für Java	Java Coding Conventions von Sun/Oracle
Sprache	Benennung von Komponenten in Diagrammen und Texten innerhalb dieser Entwicklungsdokumentation in Deutsch und selten Englisch. Verwendung englischer Bezeichner für Klassen, Methoden etc. im Java-Quelltext



3. Fachlicher Kontext

VIER. richtet sich vor allem an Studenten des Kurses WWI14SCA, da diese im Rahmen eines Turniers gegen die Applikationen der anderen Kursteilnehmer spielen können. VIER. bietet dabei die Möglichkeit sowohl gegen menschliche Gegner als auch einen KI Gegner der anderen Gruppen zu spielen (s. Abbildung 32).

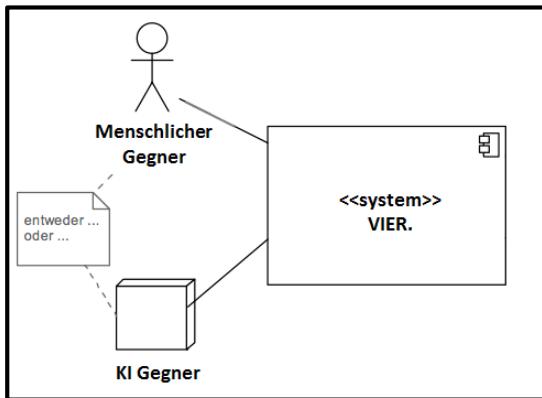


Abbildung 32: Fachlicher Kontext

3.1. Menschlicher Gegner (Benutzer)

Das Spiel „4 gewinnt“ wird zwischen zwei Gegnern gespielt, die abwechselnd ihre Steine setzen und so lange spielen, bis einer der Spieler vier Steine nebeneinander (horizontal, vertikal oder diagonal) platziieren konnte oder das Spielfeld voll ist. VIER. ermöglicht zwei Spielern das manuelle Spiel einer Partie „4 gewinnt“ an einem gemeinsamen Rechner. Dazu werden die Steine durch die Spieler in alternierender Reihenfolge auf dem Spielfeld platziert.

3.2. KI Gegner (Fremdsystem)

Alternativ zu einem menschlichen Gegner kann VIER. auch gegen eine andere Engine antreten. In diesem Modus spielt nicht der Benutzer gegen die KI, sondern VIER. ermöglicht ein automatisiertes Spiel ohne Nutzereingriff, das durch die integrierte KI geführt wird. Informationen bezüglich der gesetzten Steine werden dabei über einen zwischengeschalteten Server ausgetauscht.



4. High System Level Overview

Der „4 gewinnt“ Agent VIER. basiert auf einem mehrschichtigen Designansatz und setzt sich aus folgender Schichtenarchitektur zusammen.

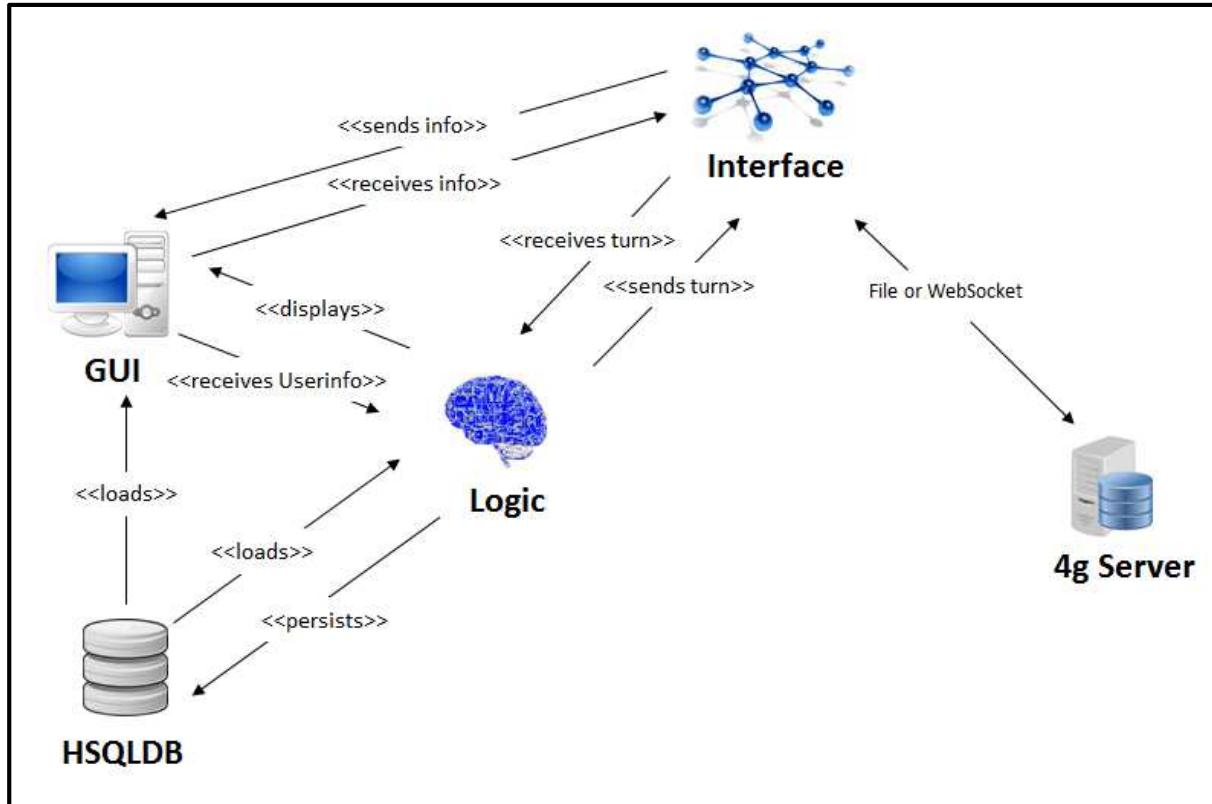


Abbildung 33: High System Level Overview

4.1. Logik

In der Logikschicht ist die KI implementiert, die ein automatisches Spiel gegen gegnerische Agenten ermöglicht. Die Berechnung des nächsten Zuges basiert dabei auf dem Alpha-Beta Algorithmus.

4.2. Schnittstellen

Die Schnittstellen ermöglichen die Kommunikation zwischen Agent und Server. Diese findet hier je nach Einstellung entweder über Dateien auf Basis einer File-Schnittstelle oder über WebSockets via Pusher-Interface statt. Die Kommunikationsschicht interagiert hierbei vor



allem mit der Logik und sorgt für den Empfang sowie die Übermittlung gespielter Züge, interagiert aber ebenfalls mit der graphischen Oberfläche, da hier zum Einen Benutzereinstellungen wie die verwendete Schnittstelle abgefragt werden, sowie zum Anderen der vom Server ermittelte Gewinner eines Satzes an die graphische Oberfläche zur Darstellung weitergeleitet wird.

4.3. Graphische Oberfläche

Die graphische Oberfläche wurde über Programmierung in JavaFX realisiert und bietet somit ein modernes Frontend, das individualisierte Einstellungen durch den Benutzer zulässt.

4.4. Datenhaltung

Die Datenhaltungsschicht kommuniziert mit der angebundenen Datenbank HSQL. Hier werden unter anderem gespielte Spiele, Punkte, Spieler sowie die Züge abgelegt, um eine Nachverfolgung bereits gespielter Spiele zu ermöglichen.

5. Produktstruktur

Die Architektur wurde in mehreren Paketen umgesetzt. Abbildung 34 gibt hier einen Überblick über die Pakete und die ihnen zugeordneten Klassen.

Die Klassen steuern dabei Teilbereiche des Programms. Die zentrale Klasse „MainApplication“ startet die Applikation.

Das Paket Interfaces ermöglicht die Interaktion des Agenten mit dem Server über eine File- bzw. eine Psuher-Schnittstelle. Weiterhin findet eine Interaktion mit der GUI sowie der Logik statt. Im Paket Logik werden Spiele, Sätze sowie Züge angelegt und die nächsten Züge mit Hilfe des Alpha-Beta-Algorithmus neu berechnet.

Das Paket DB befasst sich schließlich mit der Datenhaltung und ermöglicht gleichermaßen das Auslesen gespeicherter Daten aus der Datenbank. Über die Klasse DBConnector können die Daten in der angeschlossenen Datenbank HSQLDB abgelegt werden.

Das Paket GUI realisiert die grafische Oberfläche des Agenten und ermöglicht Ausgaben wie z.B. die Gewinnmeldung.

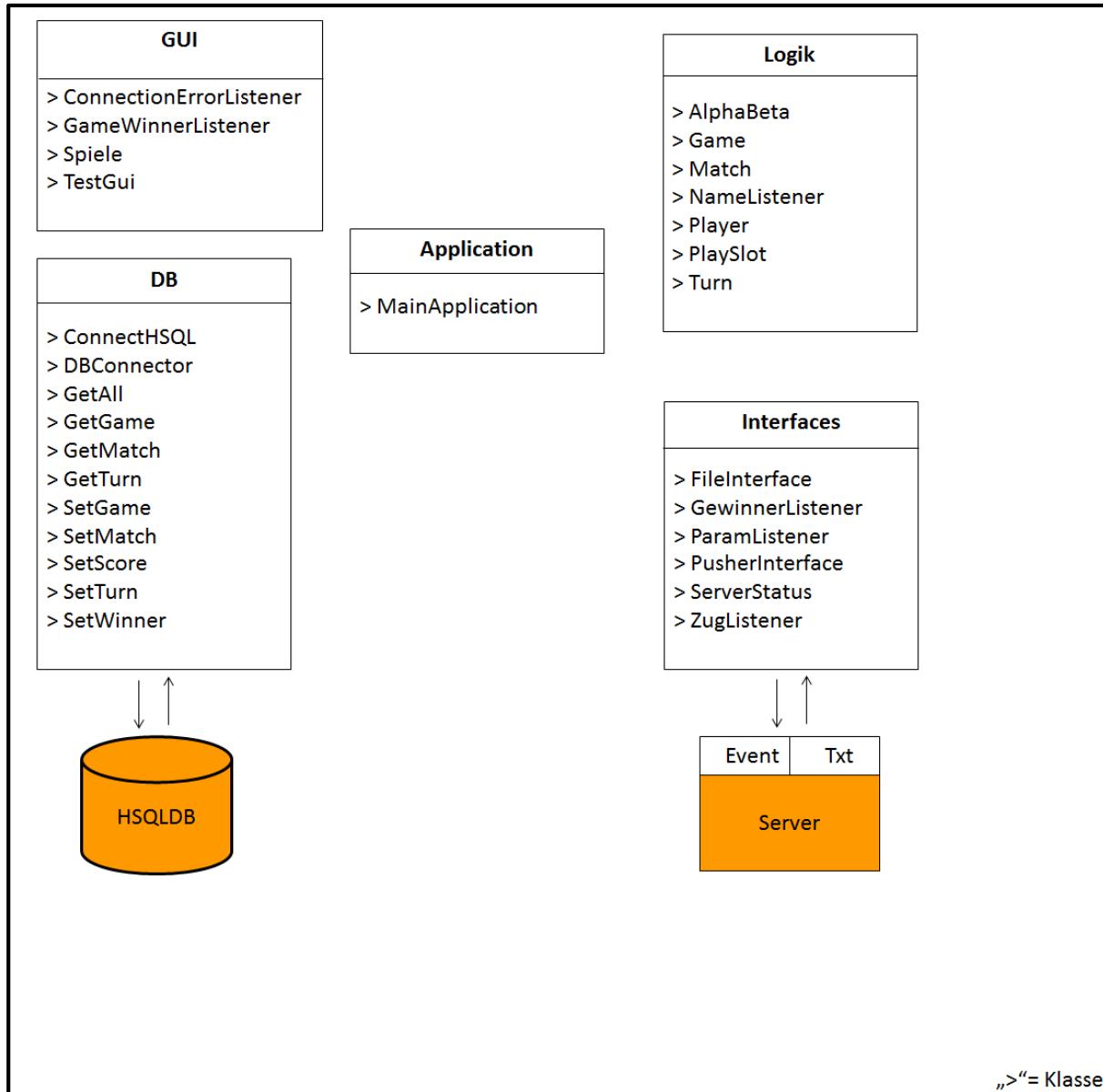


Abbildung 34: Produktstruktur VIER.

6. Fachklassen

Die Fachklassen wurden mit dem kostenfreien Eclipse Plugin ObjectAid UML Explorer angefertigt. Hierfür wurde der Java Quellcode ausgelesen und automatisch ein Diagramm erstellt. Zur besseren Übersichtlichkeit wurden Getter- und Setter-Methoden sowie EventListener nicht dargestellt.

Aufgrund der äußerst großen Komplexität des Diagramms ist es dem Projekt zusätzlich beigefügt und steht digital als JPEG zur Verfügung.

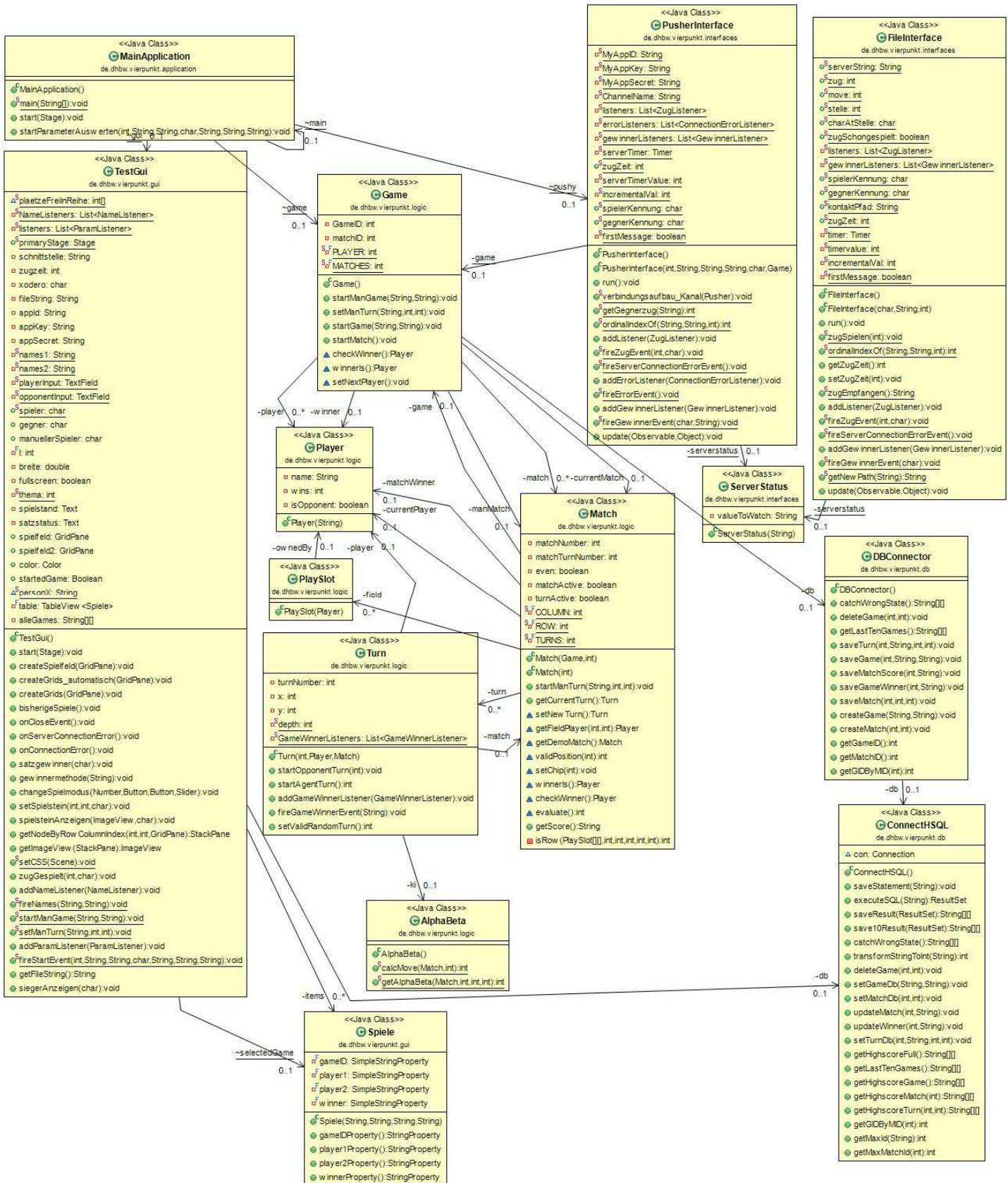


Abbildung 35: Fachklassendiagramm VIER.



7. Konzeption

7.1. Die Startklasse

In nachfolgenden Abschnitt wird die Startklasse erläutert, sowie deren wichtigsten Attribute und Funktionen erklärt. Allgemeine Methoden, wie Getter- und Setter-Methoden, finden hier keine nähere Erwähnung. Für weitere Informationen ist auf den Quellcode des Pakets `de.dhbw.vierpunkt.application` zu verweisen. Des Weiteren wird von der Beschreibung eindeutiger Attribute abgesehen.

Paket: `de.dhbw.vierpunkt.application`

Folgende Klasse ist in oben benanntem Paket enthalten:

- **MainApplication.java**

Die Klasse MainApplication.java ist die Hauptklasse der Anwendung. In ihr werden Instanzen aller Komponenten erzeugt und verwendet.

Main-Methode:

In der Main-Methode wird die Kommunikation der zuvor initialisierten Klassenobjekte ermöglicht, indem die Events-sendenden Objekte die jeweiligen Empfänger ihrer Events zu Listener-Listen hinzufügen. Für weitere Informationen zur Kommunikation zwischen den Klassen siehe: #Java-Interfaces im Package Interfaces. Anschließend wird die Start-Methode aufgerufen.

Start-Methode:

In der Start-Methode wird die PrimaryStage der MainApplication dem GUI übergeben. Damit liegt die Darstellungslogik in der GUI-Klasse, der JavaFX-Application Thread bleibt in der MainApplication.

startParameterAuswerten-Methode:

Die Methode zum Auswerten der Startparameter prüft welche Schnittstelle ausgewählt wurde und startet den entsprechenden Thread. Dabei werden die für die jeweilige Schnittstelle relevanten Informationen über Konstruktoren übergeben. Bei Auswahl des FileInterface wird zusätzlich das GUI zum Listener für ZugEvents.



7.2. Die Spiellogik

Im nachfolgenden Abschnitt werden die Klassen der Spiellogik erläutert, sowie deren wichtigsten Attribute und Funktionen erklärt. Allgemeine Methoden, wie Getter- und Setter-Methoden, finden hier keine nähere Erläuterung. Für weitere Informationen ist auf den Quellcode des Pakets `de.dhbw.vierpunkt.logic` zu verweisen. Des Weiteren wird von der Beschreibung eindeutiger Attribute abgesehen.

Paket: `de.dhbw.vierpunkt.logic`

Folgende Klassen sind in oben benanntem Paket enthalten:

- **AlphaBeta.java**
Implementierung der KI auf Basis des Alpha-Beta-Algorithmus.
- **Game.java**
Das Spielobjekt, welches drei Sätzen (Matches) entspricht.
- **Match.java**
Das Objekt, welches einen Satz abbildet.
- **NameListener.java**
Dieses Interface ist die Schnittstelle zwischen GUI und Logik.
- **Player.java**
Das Spielerobjekt, welches einen Spieler im Spiel darstellt.
- **PlaySlot.java**
Diese Klasse stellt eine Spielposition dar, welche im Satz, als Array definiert, ein Spielfeld abbildet.
- **Turn.java**
Dieses Objekt stellt einen Zug im Satz dar.

Nachfolgend werden die Klassen einzeln näher erläutert, sowie die wichtigsten Attribute und Methoden dargestellt. Nähere Erläuterungen, sowie auch Implementierungsdetails sollten der JavaDoc entnommen werden.

7.2.1. Der Algorithmus – Klasse „AlphaBeta“

Die Klasse „AlphaBeta“ hat folgende Aufgaben:

- Identifizieren des bestmöglichen Zuges auf Basis der aktuellen Spielsituation
- Maximieren des Outputs für unseren Agenten
- Minimieren des Outputs für den Gegner



Die Klasse „AlphaBeta“ entspricht einer künstlichen Intelligenz, die den bestmöglichen Zug im Spiel berechnet. Der Algorithmus beruht auf der Alpha-Beta-Suche, eine Erweiterung des sogenannten MiniMax-Algorithmus. Dieser beruht darauf, dass die möglichen Züge eines Spiels in einem Suchbaum dargestellt werden, der dann durchsucht wird. Hierbei wird zwischen MAX-Knoten (der zu optimierende Spieler ist am Zug) und MIN-Knoten (der Gegner ist am Zug) unterschieden. Die Anzahl der zu durchsuchenden Kindknoten liefert die Suchtiefe (*depth*). Die Suchtiefe hat Einfluss auf die Rechenzeit des Algorithmus. Je höher die Suchtiefe, desto länger die Rechenzeit (da dann mehr Züge berechnet werden). Die folgende Abbildung stellt beispielhaft einen Spielbaum für das Spiel „Tic Tac Toe“ dar.

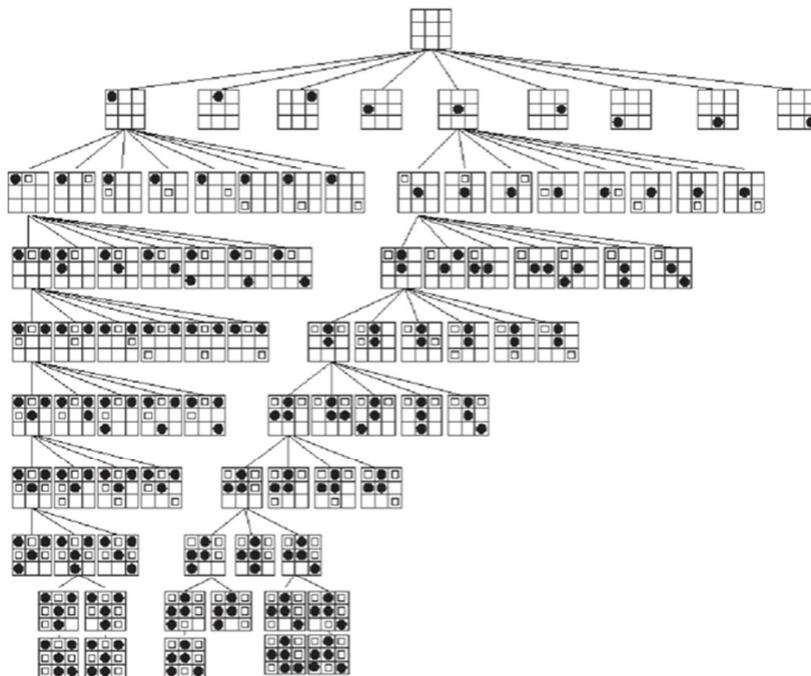


Abbildung 36: Ausschnitt eines Spielbaums von Tic Tac Toe¹

Der Alpha-Beta-Cut optimiert den MiniMax-Algorithmus nun dahingehend, dass im Schnitt weniger Knoten durchsucht werden müssen, da manche Kindknoten von vornherein ausgeschlossen werden (diese würden ohnehin schlechtere Werte liefern). Im Folgenden ist der Alpha-Beta-Schnitt schematisch dargestellt:

¹ Panitz, Sven Eric: Java will nur spielen : Programmieren lernen mit Spaß und Kreativität. 2. Aufl.. Berlin Heidelberg New York: Springer-Verlag, 2011. Seite 207.

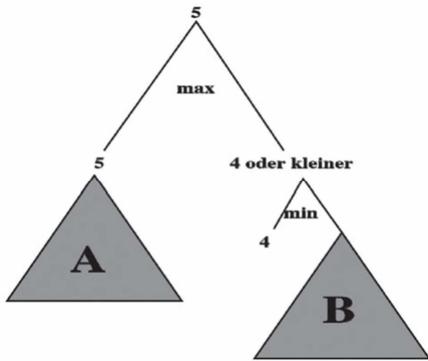


Abbildung 37: Beispiel für einen Alpha-Beta-Cut²

Für den Alpha-Beta-Cut wird der aktuell beste (für MAX) oder aktuell schlechteste (für MIN) MiniMax-Wert (*minimax_curr*) in eine Variable geschrieben (alpha bzw. beta). Falls dann erkannt wird, dass ein neuer Knoten schlechter für MAX ist als der bereits untersuchte, wird dieser erst gar nicht weiter betrachtet (*wenn alpha >= beta*). Der alte Knoten dient also als Untergrenze; es macht keinen Sinn, einen schlechteren Zug weiter zu betrachten. Analog funktioniert der Schnitt als Obergrenze für einen Zug von MIN. Wenn hier festgestellt wird, dass Zug II besser als Zug I ist (*wenn beta <= alpha*), wird Zug II nicht näher betrachtet, da er einen Nachteil für den zu optimierenden Spieler (MAX) darstellen würde. Wenn alle relevanten/ vordefinierten Züge betrachtet wurden (*depth == 0*), wird die Spielsituation bewertet, d.h. die Zugkombination wird numerisch beziffert. Dies geschieht in der Methode *evaluate()* der Klasse *Match*.

In diesem Fall ist der Alpha-Beta-Algorithmus so umgesetzt, dass zunächst aus der Klasse *Turn* heraus die Methode *calcMove(Match match, int depth)* aufgerufen wird, wenn unser Agent (der zu maximierende Spieler) am Zug ist. In dieser Methode werden alle möglichen Züge geprüft und deren Bewertung in einem int-Array (*values*) abgelegt. Um nicht das aktuelle Spielfeld zu manipulieren, werden die möglichen Zugkombinationen in einem *demo-Match* durchgespielt. Wenn der berechnete Wert der neuen Spalte (*max*) höher ist als der der vorherigen Spalte, wird diese Spalte ausgewählt (*move*) und deren Bewertung gespeichert. Dieser Vorgang wird für alle Spalten iterativ durchgeführt. Zum Schluss wird dann die bestmögliche Spalte (*move*) an die Klasse *Turn* zurückgegeben.

² Panitz, Sven Eric: Java will nur spielen : Programmieren lernen mit Spaß und Kreativität. 2. Aufl.. Berlin Heidelberg New York: Springer-Verlag, 2011. Seite 214.



Um die Bewertung für die jeweilige Spalte zu erhalten, wird die Methode `getAlphaBeta(Match match, int depth, int alpha, int beta)` aufgerufen. Wenn der aufrufende Spieler am Zug ist, wird maximiert (`!match.getCurrentPlay().getIsOpponent()`), wenn der Gegner am Zug ist, wird minimiert. Hierfür wird rekursiv jede mögliche Zugkombination überprüft, es sei denn, ein Zweig kann von vornherein abgeschnitten, d.h. ausgeschlossen werden (s.o.: Alpha-Beta-Cut). Wenn die Suche abgeschlossen ist (`depth == 0`), wird die jeweilige Zugkombination in der Methode `evaluate()` bewertet.

7.2.2. Das Spielobjekt – Klasse „Game“

Die Klasse „Game“ hat folgende Aufgaben:

- Implementierung einer Schnittstelle zur GUI
- Erstellen eines Spiels
- Verwalten der Spieler
- Verwalten der Sätze
- Speichern der Informationen in der Datenbank

In der Klasse „Game“ ist das Interface `NameListener` implementiert, dieses wird seitens der GUI angesprochen und ruft die Methode `startGame()` auf, sobald der Benutzer in der Oberfläche auf den Button („Spiel starten“) drückt. Als Parameter werden die Namen der beiden Spieler übergeben, welche über die Benutzeroberfläche bei Programmstart eingegeben wurden. Die Methode erstellt ein `player[]`-Array und hinterlegt die beiden Spieler jeweils in die Arrayposition 0 (Spieler 1) und 1 (Spieler 2). Die GUI übergibt als Spieler 2 immer den Gegner, weshalb diese Information beim Spieler hinterlegt wird. Zuletzt werden die beiden Spieler und somit das Spiel in der Datenbank angelegt (`db.createGame()`).

Ein Spiel besteht immer aus drei Sätzen, welche in einem `match[]`-Array verwaltet werden. Zum Anlegen eines neuen Satzes, wird die Methode `startMatch()` aufgerufen. Diese sucht über eine for-Schleife nach der nächsten freien Stelle im Array und legt dort ein neues Match an. Dem neuen Match wird das Spielobjekt (Game) und die Satznummer (`matchNumber`), welche der Position im `match[]`-Array entspricht, übergeben. Das Match wird in der Daten-



bank angelegt, als *currentMatch* hinterlegt und direkt ein neuer Zug mittels Aufruf der Methode *currentMatch.startNewTurn()* gestartet. Über die Boolean-Variable *matchActive* wird festgestellt, ob derzeit ein Satz aktiv ist. Sobald ein Satz beendet wurde, wird ein neuer Satz gestartet, da in einer kopfgesteuerten *while*-Schleife sowohl die *matchNumber*, *matchActive* als auch eine Gewinnsituation in einem Match abgeprüft werden.

7.2.3. Das Satzobjekt – Klasse „Match“

Die Klasse „Match“ hat folgende Aufgaben:

- Verwalten der Züge
- Verwalten des Spielfeldes
- Bewertung der gesamten Spielsituation

Der Konstruktor mit Parametern der Klasse „Match“ erwartet ein Spielobjekt und die *matchNumber*. Damit der Alpha-Beta-Algorithmus (KI) die besten Züge kalkulieren kann, wird eine Kopie des aktuellen Satzobjekts benötigt, welches über die Methode *getDemoMatch()* erstellt wird und einen abweichenden Konstruktor, welcher nur die *matchNumber* als Parameter erwartet, benötigt.

Die Züge des Satzes werden über das *turn[]*-Array verwaltet. Ein neuer Zug wird über die Methode *setNewTurn()* angelegt, indem über eine for-Schleife die nächste freie Stelle im Array gesucht, der Zug erstellt und durch diesen das *turn[]*-Objekt zurückliefert wird. Analog zu der Erstellung des Match-Objekts wird auch hier als Parameter die *turnNumber*, welche der Position im Array entspricht, übergeben. Des Weiteren werden der aktuelle Spieler und das Match-Objekt übergeben. Weitere Züge werden angelegt, sobald der Zug eines Spielers beendet ist. Um dies festzustellen, wird analog zur Klasse „Game“ auch hier eine *turnActive*-Variable verwendet. Diese Boolean-Variable wird auf „TRUE“ gesetzt, sobald ein neuer Zug angelegt wurde. Damit die Methode *getCurrentTurn()* immer das korrekte Zugobjekt über gibt, wird die aktuelle *turnNumber* (Position des Arrays) abgespeichert.

Das Spielfeld wird über das *field[y][x]*-Array der Klasse „PlaySlot“ verwaltet. Dieses wird beim Erstellen eines Satzes initialisiert und hinterlegt den Spieler, welcher in die Position gespielt



hat. Hierfür werden die Methoden `setField()` bzw. `getFieldPlayer()` benötigt. Das Array entspricht der Größe des Spielfeldes, wobei die Stelle x der Spalte und die Stelle y der Zeile im Spielfeld entspricht. Um die nächste gültige Position in einer Spalte zu bestimmen, wird die Methode `validPosition()` verwendet. Diese erwartet als Parameter die Spalte x und liefert nach Prüfung im Spielfeld die nächste gültige Zeile y als Rückgabewert. Kann keine gültige Position gefunden werden, liefert die Methode den Wert -1.

Um die Bewertung der Spielsituation vornehmen zu können, greift die Methode `isRow()` auf das Spielfeld zu, um die Anzahl der „Steine“ eines Spielers (cnt) zu zählen und zurückzugeben. `isRow()` prüft pro Viererreihe nicht nur, ob Steine hintereinander, sondern auch, ob zwischendrin freie Felder liegen. Im nachfolgenden Beispiel würde der Count des Spielers also sowohl in Abbildung 38, als auch in Abbildung 39 entsprechen.



Abbildung 38: Beispiel Berechnung `isRow()`



Abbildung 39: Beispiel Berechnung `isRow()`

Mit Hilfe des zurückgelieferten Wertes kann dann über die Methode `evaluate()` berechnet werden, wie viele 2er, 3er oder 4er-Kombinationen ein Spieler aktuell auf dem Spielfeld belegt hat. Diese Prüfung findet horizontal, vertikal und diagonal (nach rechts und nach links) statt. Die Kombinationen werden dann unterschiedlich stark gewichtet (eine 3er-Kombination des Gegners ist unbedingt zu verhindern, eine 3er-Kombination unseres Spielers ist anzustreben) und als Zahlenwert zurückgegeben (`eval`). Dieser Zahlenwert gibt Orientierung darüber, ob der Spieler in einer positiven oder negativen Spielsituation ist. Wird eine positiv unendliche Zahl geliefert, ist das Spiel gewonnen, bei einer negativ unendlichen Zahl liegt eine Verlierersituation vor.



7.2.4. Die Schnittstelle – Klasse „NameListener“

Das Interface „NameListener“ hat folgende Aufgaben:

- Übergeben der Spielernamen aus der GUI
- Aufruf der Methode `startGame()` in der Klasse „Game“

Um erfassen zu können, wenn der Benutzer den Button „Spiel starten“ ausgewählt hat und damit die durch den Benutzer eingegeben Spielernamen zu erhalten, wird ein Interface benötigt. Die GUI übergibt die Namen an das Interface „NameListener“, welches wiederum die Methode `startGame()` in der Klasse „Game“ aufruft. Da kein direkter Zugriff seitens der Logik zur GUI und umgekehrt möglich ist, ist diese Schnittstellenklasse erforderlich.

7.2.5. Das Spielerobjekt – Klasse „Player“

Die Klasse „Player“ hat folgende Aufgaben:

- Anlegen eines neuen Spielerobjekts
- Verwalten des Spielerobjekts

Beim Anlegen eines neuen Spielers wird die Klasse „Player“ angesprochen. Im Konstruktor erwartet die Klasse „Player“ den Spielernamen als „STRING“. Dieser wird als Attribut *name* hinterlegt, sowie die sonstigen Attribute initialisiert.

Ein Spieler kann ein Spiel gewinnen (*wins*). Der Zahlenwert wird benötigt, um den Spielstand (Bsp.: 1:0, etc.) darstellen zu können. Innerhalb der Klasse „Match“ wird dieser als „STRING“ zusammengesetzt und über die Klasse „Turn“ auf Ebene der Klasse „Game“ in die Datenbank gespeichert. Des Weiteren kann ein Spieler ebenfalls durch den Gegner besetzt sein. Ist der Spieler der Gegner, wird die Boolean-Variable *isOpponent* auf „TRUE“ gesetzt.



7.2.6. Eine Spielposition – Klasse „PlaySlot“

Die Klasse „PlaySlot“ hat folgende Aufgabe:

- Anlegen einer Spielposition

Beim Anlegen einer Spielposition, die in der Klasse „Match“ benötigt wird, um das Spielfeld darzustellen, wird der Spieler, welcher in die Position „geworfen“ hat, als Parameter übergeben und im Attribut *ownedby* gespeichert. Hat das Attribut den Wert „NULL“, ist kein Spielerobjekt hinterlegt und die Position somit frei.

7.2.7. Der Spielzug – Klasse „Turn“

Die Klasse „Turn“ hat folgende Aufgaben:

- Anlegen eines Spielzuges
- Verwalten eines Spielzuges

Beim Anlegen eines Spielzuges wird eine eindeutige Zugnummer (*turnNumber*), der Spieler, der den Zug durchführt, sowie das Satzobjekt als Parameter erwartet.

Beim Spielen über das Pusher- und / oder das File-Interface wird ein Zug durch den Agenten (Spieler seitens des Clients) bzw. durch den Opponent (Gegner) erwartet. Die Methode *startAgentTurn()* erwartet keine Parameter und kalkuliert den Zug über den Alpha-Beta-Algorithmus. Die Methode *calcMove()* aus der Klasse „AlphaBeta“ erwartet als Paramter das Satzobjekt und die Suchtiefe. Je höher die Suchtiefe eingestellt ist, desto weiter kalkuliert der Algorithmus den Zug, was wiederum Auswirkungen auf die Performance haben kann. Als Rückgabewert liefert *calcMove()* die Spalte *x*, in welche geworfen werden soll. Durch den Aufruf der Methode *validPosition()* der Klasse „Match“ wird die Zeile *y* ermittelt. Anschließend wird der Zug über die Methode *setField()* der Klasse „Match“ ausgeführt und dieser in der Datenbank abgelegt. Gefolgt ist dieser Aufruf vom Prüfen, ob ein Gewinner im Satz vorliegt, das Setzen des nächsten Spielers und das Ändern der *TurnActive*-Variable der Klasse „Match“ auf den Wert „FALSE“. Zuletzt wird ein neuer Zug über den Aufruf der Methode *startNewTurn()* der Klasse „Match“ initiiert. Die Methode *startAgentTurn()* liefert als Rückgabewert die Spalte *x*.



Die Methode `startOpponentTurn()` erwartet die Spalte x , in welche der Gegner eingeworfen hat. Der Zug wird nicht kalkuliert, sondern direkt die gültige Zeile ermittelt. Analog zur Methode `startAgentTurn()` werden alle oben bereits genannten Schritte durchgeführt. Die Methode `startOpponentTurn()` liefert keinen Rückgabewert.

7.3. Schnittstellen

Im nachfolgenden Abschnitt werden die Klassen der Schnittstellen erläutert, sowie deren wichtigsten Attribute und Funktionen erklärt. Allgemeine Methoden, wie Getter- und Setter-Methoden, finden hier keine nähere Erläuterung. Für weitere Informationen ist auf den Quellcode des Pakets `de.dhbw.vierpunkt.interfaces` zu verweisen. Des Weiteren wird von der Beschreibung eindeutiger Attribute abgesehen.

Paket: de.dhbw.vierpunkt.interfaces

Folgende Klassen sind in oben benanntem Paket enthalten:

- **FileInterface**
Erlaubt die Kommunikation mit dem Server auf Basis von Streams
- **GewinnerListener**
Hilft bei der Darstellung des Siegers eines Satzes auf der GUI
- **ParamListener**
Hilft dabei Einstellungen wie verwendete Schnittstelle, Zugzeit, Kontaktlauf, Spielerkennung zu empfangen
- **PusherInterface**
Ermöglicht die Kommunikation mit dem Server auf Basis von WebSockets
- **ServerStatus**
Dient der Überwachung, ob der Server noch aktiv ist
- **ZugListener**
Dient der Weitergabe der empfangenen Züge an die GUI

Nachfolgend werden vor allem die Klassen FileInterface sowie PusherInterface einzeln näher erläutert, sowie die wichtigsten Attribute und Methoden dargestellt. Nähere Erläuterungen, sowie auch Implementierungsdetails sollten der JavaDoc entnommen werden.



7.3.1. Klasse PusherInterface.java

Die Klasse PusherInterface realisiert die Verbindung zwischen der Pusher-Instanz des Servers und der GUI, sowie der Spiellogik. Dabei werden zunächst in einem Konstruktor die Werte AppID, AppKey und AppSecret übergeben, die für einen Verbindungsaufbau zum Pusher Server benötigt werden. Außerdem werden Zugzeit (int in Millisec) und Spielerkennung („x“ oder „o“) übergeben. Diese Werte werden im GUI vor dem Start des Spiels festgelegt und bei betätigen des „Spiel starten“ Buttons übergeben.

In der *Run*-Methode wird, um auf einen Ausfall des Spielserver reagieren zu können, ein Timer implementiert, der eine festgelegte Zeit läuft und bei Ablauf das Spiel unterbricht. Bei jedem Event, das durch den Server an die lokale Pusher-Instanz versendet wird, wird der Timer zurückgesetzt, sodass nur durch eine ungewöhnlich lange Inaktivität eine Unterbrechung ausgelöst wird.

Um die lokale Pusher-Instanz zu autorisieren, wird in der *Run*-Methode zunächst ein Authorizer-Objekt mit den App-Werten erzeugt. Dieses erzeugt wiederum mit *signature* den String, der für die Authentisierung verwendet wird. Danach ist es der lokalen Pusher-Instanz möglich, sich an den Kanälen der Server Instanz anzumelden und die dort versendeten Events zu empfangen und zu verarbeiten. Für genauere Betrachtung der Pusher Verbindung sei auf die Pusher API verwiesen:

<https://github.com/pusher/pusher-websocket-java#api-overview>

In der Methode *Verbindungsauftbau_Kanal* wird anschließend die lokale Pusher-Instanz am privaten Kanal „private-channel“ angemeldet und ist somit befugt, dessen Events zu empfangen und Client-Events zu triggern. In diesem Fall wird auf Events mit dem Namen „MoveToAgent“ reagiert, dabei wird die in den Anforderungen dokumentierte Serverfile empfangen und ausgelesen. Danach wird durch den Aufruf von Spiellogik und Datenbank der gegnerische Zug gespeichert und ein eigener Zug berechnet. Dieser wird mit dem Client-Event „client-event“ an den Pusher Server versendet, der diesen an den Spiel-Server weitergibt.

Um Auf einen Ausfall des Spielserver reagieren zu können, wurde ein Timer implementiert, der eine festgelegte Zeit läuft und bei Ablauf das Spiel unterbricht. Bei jedem Event, das durch den Server an die lokale Pusher-Instanz versendet wird, wird der Timer zurückgesetzt, sodass nur durch eine ungewöhnlich lange Inaktivität eine Unterbrechung ausgelöst wird.



Die Methode *GetGegnerzug* nimmt den vom Server empfangenen String entgegen und zieht die Spalte, in die der gegnerische Spielstein geworfen wurde heraus.

In der Methode *OrdinalIndexOf* wird ein übergebener String auf das n-te vorkommen vom String s überprüft, es wird die Position des Strings als Integer zurückgegeben.

Darauf folgen mehrere Methoden, die die Kommunikation zwischen dem Pusher-Interface und den anderen Klassen der Anwendung realisieren. Für mehr Informationen sei an dieser Stelle auf den Punkt *Java-Interfaces im Paket Interfaces* verwiesen.

7.3.2. Klasse FileInterface.java

Die Klasse FileInterface realisiert die Verbindung zwischen der Anwendung und dem Spielsever, indem auf zwei geteilte Dateien zugegriffen wird. Dabei wird die Datei Server2spieler'x'/'o'.xml für die Kommunikation in Richtung des Spielers verwendet. In dieser Datei sind die Informationen Freigabe, Satzstatus, Zug und Sieger enthalten. In der Methode *zugEmpfangen* wird der Inhalt in dieser Datei in einem String gespeichert.

Für die entgegengesetzte Richtung wird die Datei Spieler'x'/'o'2Server.xml verwendet. Hier wird von der Anwendungsseite lediglich die Spalte übergeben, in die der Spielstein eingeschrieben werden soll. Die Datei wird von der Methode *zugSpielen* beschrieben, der ein Integer zwischen 0 und 6 übergeben wird.

Auch hier wurde, wie in der Klasse PusherInterface, ein Timer implementiert, um auf einen Ausfall des Spielerservers reagieren zu können. Der Timer läuft eine festgelegte Zeit und unterbricht bei Ablauf das Spiel. Bei jeder Änderung, die vom Server an der Server2spieler-Datei vorgenommen wird, wird der Timer zurückgesetzt, sodass nur durch eine ungewöhnlich lange Inaktivität eine Unterbrechung ausgelöst wird.

In der Methode *OrdinalIndexOf* wird ein übergebener String auf das n-te vorkommen vom String s überprüft, es wird die Position des Strings als Integer zurückgegeben.

Ebenfalls wurde wie in der Klasse PusherInterface die Kommunikation mit den Klassen von GUI und Logik auf Grundlage primitiver Events realisiert. Für mehr Informationen sei an dieser Stelle auf den Punkt *Java-Interfaces im Package Interfaces* verwiesen.



7.3.3. Interfaces im Package Interfaces

Die Kommunikation zwischen den Interfaces und den anderen Klassen sind auf Grundlage primitiver Events realisiert. Diese folgen stets dem folgenden Schema: Zunächst wird ein Java-Interface erstellt, das die Event-empfangende Klasse implementieren muss. In den Methoden in diesem Interface werden die zu übergebenden Parameter festgelegt. Anschließend wird in der Event-sendenden Klasse eine Liste mit den Listenern erstellt, die das Interface implementieren. Mit der Notation `fireEvent()` wird die Liste durchschritten und es werden jeweils die im Interface implementierten Methoden der Event-empfangenden Klassen aufgerufen.

7.4. Gui

In nachfolgenden Abschnitt werden die Klassen der GUI erläutert, sowie deren wichtigsten Attribute und Funktionen erklärt. Allgemeine Methoden, wie Getter- und Setter-Methoden, finden hier keine nähere Erläuterung. Für weitere Informationen ist auf den Quellcode des Pakets `de.dhbw.vierpunkt.gui` zu verweisen. Des Weiteren wird von der Beschreibung eindeutiger Attribute abgesehen.

Paket: `de.dhbw.vierpunkt.gui`

Die Gui besteht aus insgesamt sieben Hauptklassen.

- `Gui.java`
- `Spiele.java`
- `Gui.css`
- `Halloween.css`
- `Sweets.css`
- `Food.css`
- `Sport.css`

Der Aufbau mit den verschiedenen css-Dateien ist notwendig, da die GUI mehrere Themen zu Auswahl bietet, die zu Farb- und Symboländerungen führen. Damit sowohl die Hauptstange als auch untergeordnete Stages und Popups einheitlich gestaltet sind, werden hierfür die diversen css-Klassen verwendet.



Nachfolgend werden die Klassen einzeln näher erläutert, sowie die wichtigsten Attribute und Methoden dargestellt. Nähere Erläuterungen, sowie auch Implementierungsdetails sollten der JavaDoc entnommen werden.

7.4.1. Stages

Die Klasse Gui.java besteht aus mehreren Stages, die aus verschiedenen Aktionen heraus aufgerufen werden. Es wird zwischen folgenden Stages unterschieden:

<i>loginStage</i>	Anfangsstage
<i>primaryStage</i>	Hauptstage, von der alle weiteren ausgeht
<i>spieleStage</i>	Stage, die bisherige Spiele anzeigt

Methoden, die den Benutzer informieren (jeweils über Popup-Stages):

<i>gewinnermethode()</i>	Spieler x hat das Spiel gewonnen
<i>satzgewinner()</i>	Spieler x hat den Satz gewonnen
<i>onConnectionError()</i>	Ein Fehler bei der Pusher Connection ist aufgetreten
<i>onServerConnectionError()</i>	Ein Fehler bei der Server Connection ist aufgetreten
<i>changeSpielmodus()</i>	Das Spiel wird beendet, wenn das Thema gewechselt wird
<i>onCloseEvent()</i>	Das Fenster wird geschlossen und das Spiel abgebrochen

Zunächst öffnet sich die loginStage, welche den Benutzer nach Spielernamen fragt. Sind die Spielernamen eingegeben, öffnet sich die primaryStage, die Hauptstage der GUI. Von hier aus sind diverse weitere Stages erreichbar.

Diese Stages werden zum Teil in der Methode *start()* implementiert und durch die oben angegebenen Methoden aufgerufen. Buttons in den Stages erzeugen dann weitere Aktionen oder schließen die Stage wieder, sodass der Benutzer zu dem vorherigen Screen zurückkehrt. Stages wie die primaryStage und die spieleStage werden im Vollbildmodus angezeigt. Dies kann schnell mit der Klassenvariable Boolean fullscreen geändert werden.

Alle weiteren lediglich in der entsprechend angegebenen Größe, damit der Effekt eines Popups entsteht.



7.4.2. Layout

Alle verschiedenen Stages sind nach einem gleichen Muster aufgebaut. Dieses lässt sich besonders gut an der primaryStage erklären.

V- und HBoxes sind Elemente, die wie Container fungieren und angeben, ob deren Elemente horizontal oder vertikal angeordnet werden. Elemente die in einer HBox enthalten sind, werden deshalb nebeneinander angeordnet und Elemente die in einer VBox enthalten sind, werden vertikal angeordnet. Auf diesem Konzept basierend sind alle Elemente in der primaryStage durch Verschachtelungen von HBoxen und VBoxen angeordnet.

PopUps und weitere Stages sind auch durch solche Verschachtelungen gekennzeichnet.

Der Aufbau der primaryStage besteht aus einer VBox, welche die gesamte Stage ausfüllt. Dies bewirkt, dass alle Elemente, die in der VBox enthalten sind, untereinander angeordnet werden. In der VBox enthalten sind dann die Menuleiste, eine HBox für die Überschrift und eine HBox für den restlichen Inhalt der Stage.

Die Verschachtelungen ziehen sich so durch das Layout der GUI, sodass alle Elemente am gewünschten Platz angeordnet werden. Eigenschaften wie Alignment oder Padding ändern dann noch die Anordnung der Elemente in den Boxen, nicht aber deren Reihenfolge.

Damit sich die einzelnen Elemente einer Box nicht so dicht aneinander befinden und einen gewünschten Abstand einhalten, sind einige transparente Rectangles als Platzhalter eingefügt. Das nachfolgende Bild zeigt die angeordneten Elemente der primaryStage, aus Gründen der Übersichtlichkeit nicht mit allen eingefügten Platzhaltern.

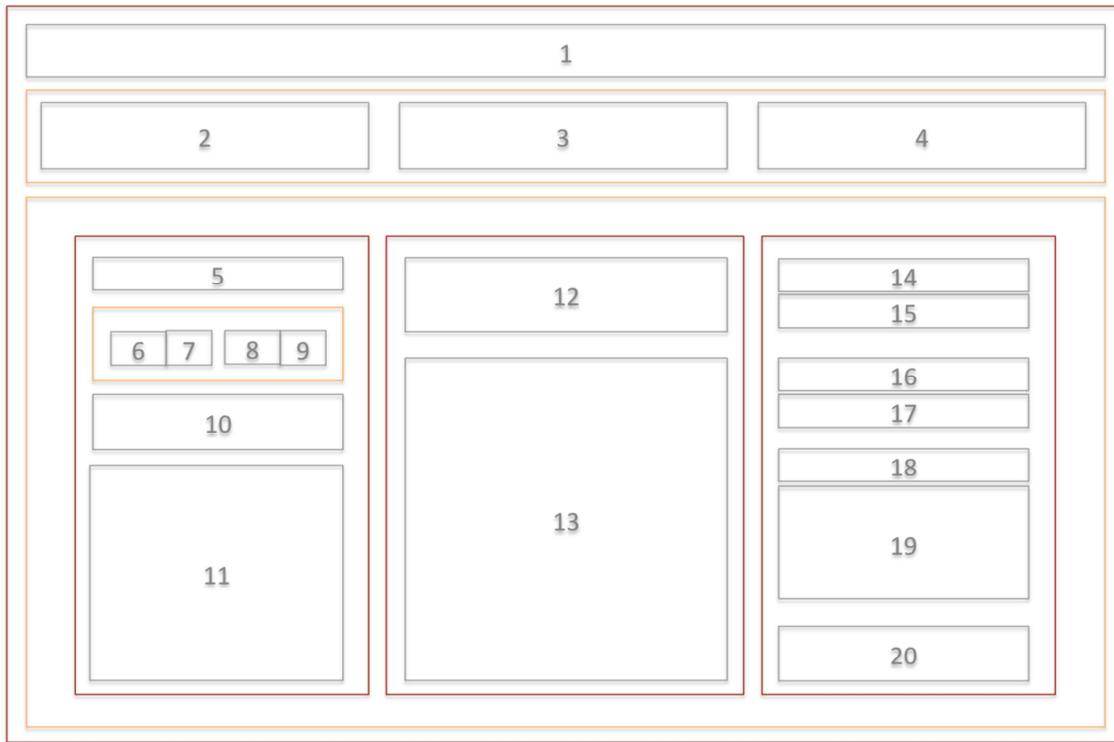


Abbildung 39: Aufbau der GUI

1	MenuBar	10	Rectangle als Platzhalter
2	Rectangle als Platzhalter	11	GridPane Spielfeld
3	Label „Spieler“	12	Label „Spielstand“
4	Label Name des Spielers	13	Text zur Anzeige des Spielstands
5	ImageView Spielstein des Spielers	14	Label „Satzstatus“
6	Label Name des Gegners	15	Text zur Anzeige des Satzstatus
7	ImageView Spielstein des Gegners	16	Label „Spielmodus“
8	Button „Einstellungen“	17	Slider zur Anzeige des Spielmodus
9	ImageView je nach aktuellem Thema	18	Button „Spiel starten“

7.4.3. EventHandler

Um nun die einzelnen Stages aufzurufen, dienen EventHandler. Diese EventHandler reagieren entweder auf MouseEvents oder sind ChangeListener und warten darauf, dass sich eine bestimmte Variable ändert.



```
// Bei Änderung des Spielstandes, wird der neue Text (t1) angezeigt  
spielstand.textProperty().addListener(new ChangeListener<String>() {  
  
    // Bei Änderung des Satzstatus, wird der neue Text (t1) angezeigt  
    satzstatus.textProperty().addListener(new ChangeListener<String>() {  
  
        // Bei Änderung des Spielmodus werden verschiedene Methoden in changeSpielmodus aufgerufen  
        spielmodus.valueProperty().addListener(new ChangeListener<Number>() {  
  
            // Bei Änderung des der Zugzeit, wird die neue Zugzeit angezeigt und durch Start mit uebergeben  
            zeit.valueProperty().addListener(new ChangeListener<Number>() {  
  
                // Bei Änderung des ToggleButtons, ändert sich der Wert des eigenen Spielers von x zu o oder andersherum  
                group.selectedToggleProperty().addListener(new ChangeListener<Toggle>() {
```

Abbildung 40: Auszug Coding ChangeListener

Dies sind die ChangeListener der Gui.java Klasse. Spielstand und Satzstatus ändern sich dann in der Gui, wenn die Variablen verändert werden. Auch bei Einstellen der Slider Spielmodus und Zeit (Zugzeit) wird auf eine Änderung gewartet. Group stellt hier unter den Einstellungen die Auswahl X oder O dar.

Ohne diese ChangeListener würde bei Änderungen der Werte nichts geschehen. Dies ist aber von enormer Bedeutung, da der Spielmodus beispielsweise angibt, ob man automatisch oder manuell spielt und damit verbunden auch, ob in die Felder des Spielfeld per Mouseclick Spielsteine gesetzt werden können oder nicht.

```
// Bei Klicken auf die Pusher Checkbox erscheinen die Credentials Felder  
pusher.setOnMouseClicked(new EventHandler<MouseEvent>() {  
  
    // Bei Klicken auf die File Checkbox erscheind der FileChooser  
    file.setOnMouseClicked(new EventHandler<MouseEvent>() {  
  
        // Bei Clicken auf Einstellungen erscheint ein Popup  
        einstellungen.setOnMouseClicked(new EventHandler<MouseEvent>() {  
  
            // bei Clicken auf Start beginnt der Verbindungsaufbau und das Spiel startet  
            start.setOnMouseClicked(new EventHandler<MouseEvent>() {
```

Abbildung 41: Auszug Coding EventHandler

Diese MouseEvents sind vor Allem bei Buttons relevant. Sobald der start („Spiel starten“) Button geklickt wird, beginnt ein neues Spiel und im automatischen Spielmodus die Kommunikation mit dem Server. Ist ein noch nicht beendetes Spiel in der Datenbank vorhanden, wird durch das Klicken auf start eine neue Stage erzeugt, welche dem Benutzer die Wahl lässt, das Spiel weiterzuspielen.



Bei dem Klick auf den Einstellungen Button wird dann die Stage angezeigt, in der man alle notwendigen Einstellungen für die Kommunikation mit dem Server tätigen kann.

Die Events die bei Pusher und File auftreten, sagen lediglich aus, welche der beiden Checkboxen gerade aktiviert ist. Sie schließen sich gegenseitig aus, das heißt es kann entweder Pusher oder File aktiviert sein. Wird File aktiviert, öffnet sich ein Fenster, in dem man den Ordner für die File-Schnittstelle angeben kann. Wird dagegen Pusher aktiviert, erscheinen weitere Einstellungsmöglichkeiten.

```
***** LOGIN BEI ENTER UND CLICK *****
playerInput.setOnKeyPressed(new EventHandler<KeyEvent>() { ...
    opponentInput.setOnKeyPressed(new EventHandler<KeyEvent>() { ...
        login.setOnKeyPressed(new EventHandler<KeyEvent>() { ...
            login.setOnMouseClicked(new EventHandler<MouseEvent>() { ...
```

Abbildung 42: Auszug Coding Login

Eine weitere Art von EventHandlern gibt es bei der loginStage, welche erscheint, sobald man das Programm startet. Hier ist es wichtig, dass man den Button nicht nur klicken kann, sondern auch auf eine Tastaturaufgabe reagiert. Egal in welchem Feld man sich gerade befindet, die Hauptstage soll aufgerufen werden, wenn man ENTER drückt. Dazu dienen KeyEventHandler, welche für die Eingabefelder playerInput und opponentInput, also auch für den Login Button gelten. In diesen EventHandlern wird zudem geprüft, ob beide Namen eingegeben wurden und dass diese nicht identisch sind. In diesen beiden Fällen werden Meldungen für den Benutzer sichtbar ausgegeben und der Login wird verweigert.

7.4.4. Das Spielfeld

Das Spielfeld, egal ob in der primaryStage oder in der spieleStage, ist das Hauptelement der GUI. Es besteht aus einer GridPane spielfeld mit sieben Spalten und sechs Zeilen.

In jeder dieser Zellen liegt eine Shape “cell” bestehend aus einem Quadrat, aus dem ein Kreis in der Mitte ausgeschnitten wurde. Dies dient zur schöneren Darstellung des Spielfeldes. Die Farbe wird je nach ausgewähltem Thema und der dazugehörigen css-Datei bestimmt.



Zusätzlich liegt in jeder dieser Zellen ein Stack. In diesem Stack werden die Spielsteine und Vorschauspielsteine abgelegt. Die Vorschauspielsteine sehen genau aus wie die Spielsteine, sind aber halbtransparent. Sie erscheinen, sobald man mit der Maus über die Zelle fährt, in die man den Spielstein legen will. Oberhalb der GridPane werden die tatsächlichen Spielsteine angezeigt, die dann nach unten in die entsprechende Zelle fallen, wenn man auf die Zelle oder den Spielstein klickt.

All diese Funktionen und auch die GridPane selbst, werden durch die Methode *createGrids()* erzeugt. Sie dient genau dann, wenn manuell gespielt werden soll. Soll aber automatisch gespielt werden, das heißt KI gegen den Server, braucht man die MouseEvents nicht. Im Gegenteil, die MouseEvents sollen dann nicht funktionieren. Hierzu erzeugt man die GridPane mit der Methode *createGrids_automatisch()*. Sie beinhaltet das Gleiche wie die Methode *createGrids()*, außer die MouseEventHandler.

```
***** Erzeugt die Zeilen und Spalten der GridPane *****/
public void createSpielfeld(GridPane spielfeldGrid){}

***** Belegung der GridPane mit Feldelementen *****/
public void createGrids_automatisch(GridPane spielfeld){}

***** Belegung der GridPane mit Feldelementen und MouseEvents *****/
public void createGrids(GridPane spielfeld){}
```

Abbildung 43: Auszug Coding GridPane

Die Methode *createSpielfeld()* ist hierbei nur eine Hilfsmethode und erzeugt die Gridpane mit der richtigen Anzahl an Spalten und Zeilen. Dies erspart Redundanzen in der Programmierung.



Zu dem Spielfeld gehören neben der Erzeugung aber auch die Methoden, die die Spielsteine tatsächlich in der GridPane anzeigen lassen. Diese Funktionen bieten die Folgenden:

```
***** Setzen des Spielsteins in der Farbe des Spielers *****
public void setSpielstein(int zeile, int spalte, char amZug){}

***** Anzeige des Spielsteins in der richtige Farbe *****
public void spielsteinAnzeigen(ImageView spielstein, char amZug){}

***** Zugriff auf die Elemente in der GridPane an der jeweiligen Stelle *****
public StackPane getNodeByRowColumnIndex (final int row, final int column, GridPane spielfeld) {}

***** ImageView des ausgewählten Spielsteins zum veraendern des Spielsteinsymbols *****
public ImageView getImageView (StackPane stack) throws NullPointerException {}
```

Abbildung 44: Auszug Coding Methoden auf GridPane

Die Methode `setSpielstein()` wird aufgerufen, wenn ein Spielstein in eine bestimmte Zelle gesetzt werden soll. Dazu werden Zeile und Spalte und für die Bestimmung der Spielsteinfarbe noch der Spieler übergeben.

`setSpielstein()` ruft wiederum die Methode `spielsteinAnzeigen()` auf, welche jedoch den entsprechenden Spielstein sowie die Spieler benötigt. Um den richtigen Spielstein in der Grid anzusprechen, liefert die Methode `getNodeByRowColumnIndex()` die StackPane an der entsprechenden Stelle in der GridPane zurück. Mit Hilfe von `getImageView()` wird dann aus diesem Stack die entsprechende ImageView zurückgegeben und `spielsteinAnzeigen()` kann diese ImageView nun verwenden.

So wird durch die Verschachtelung dieser Methoden der richtige Spielstein in der GridPane gesetzt. Andere Klassen müssen dabei lediglich die Methode `setSpielstein()` aufrufen.

7.4.5. Das Menü

Die Menüleiste ganz oben in der primaryStage ist zur Navigation sehr wichtig. Über sie lässt sich das Spiel beenden, ein neues Spiel starten oder die Themen wechseln. Das Menü ist wie eine Baumstruktur aufgebaut.

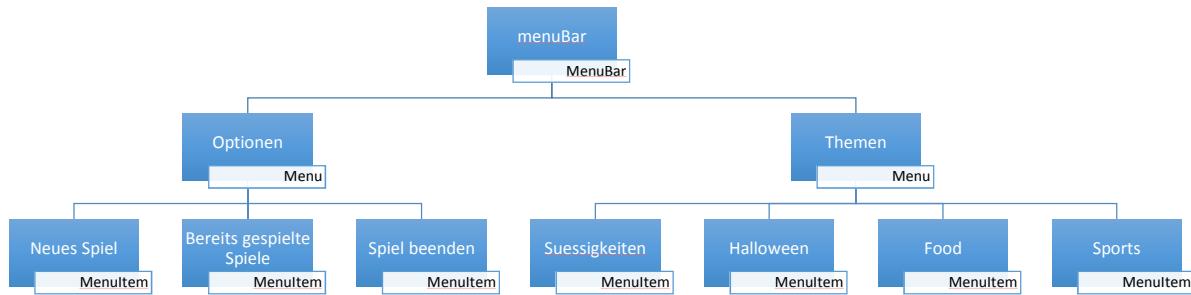


Abbildung 45: Aufbau Menü

Durck Klicken auf die MenuItem's werden dann neue Stages aufgerufen oder bei den Themen verschiedene css-Dateien ausgewählt und somit die Darstellung geändert.

7.5. Datenbank

Im nachfolgenden Abschnitt werden die Klassen der Datenbank erläutert, sowie deren wichtigsten Attribute und Funktionen erklärt.

Paket: **de.dhbw.vierpunkt.db**

Nachfolgende Klassen befinden sich im oben genannten Packet:

- **ConnectMySQL.java**
In dieser Klasse wird eine Verbindung zur Datenbank definiert und grundsätzliche Methoden zum Umgang mit dieser implementiert.
- **DBConnector.java**
Schnittstellenklasse zwischen Datenbank und dem restlichen Programm.
- **GetAll.java**
In Thread ausgelagerte Get- Methode.
- **GetGame.java**
In Thread ausgelagerte Get- Methode.
- **GetMatch.java**
In Thread ausgelagerte Get- Methode.
- **GetTurn.java**
In Thread ausgelagerte Get- Methode.
- **LegacyMethods.java**
Klasse in den veralteten Methoden zwischengespeichert werden, falls Bedarf entsteht sollte. *Kann ignoriert werden.*
- **SetGame.java**
In Thread ausgelagerte Set- Methode.



- **SetMatch.java**
In Thread ausgelagerte Set- Methode.
- **SetScore.java**
In Thread ausgelagerte Set- Methode.
- **SetTurn.java**
In Thread ausgelagerte Set- Methode.
- **SetWinner.java**
In Thread ausgelagerte Set- Methode.

Nachfolgend werden die Klassen einzeln näher erläutert, sowie die wichtigsten Attribute und Methoden dargestellt. Nähere Erläuterungen, sowie auch Implementierungsdetails, können der JavaDoc entnommen werden.

7.5.1. ConnectHSQL.java

In der ConnectHSQL.java wird ein Connection Objekt erstellt, welche alle Verbindungsdaten für die Datenbank beinhaltet. Falls es zu einem SQL Error kommt, wird eine SQL Exception ausgegeben. In der Klasse wird hauptsächlich mit drei Methoden gearbeitet.

public void saveStatement(String sql)

Diese Methode übernimmt ein SQL Statement und übergibt es der Datenbank. Außerdem hat diese Methode keinen Rückgabeparameter. Falls das SQL Statement fehlerhaft ist, wird eine SQL Exception ausgegeben. Diese Methode eignet sich somit nur für Einwegkommunikation.

public ResultSet executeSQL(String sql)

Diese Methode übernimmt ein SQL Statement und übergibt es der Datenbank. Das Ergebnis wird in einem Resultstatement gespeichert und zurückgegeben. Falls das SQL Statement fehlerhaft ist, wird eine SQL Exception ausgegeben.

public String[][] saveResult(ResultSet result)

Diese Methode übernimmt ein ResultSet und fügt es mittels einer Schleife zu einem String-Array zusammen. Falls der Inhalt des ResultSets fehlerhaft ist, wird eine SQL Exception geworfen.



public String[][] save10Result(ResultSet result)

Spezielle Erweiterung der obigen Methode um ein Array, genau der Zeilengröße 10 zu erstellen. Alle anderen Funktionen lassen sich aus der Methode saveResult ableiten.

public String[][] checkWrongState()

Diese Methode testet ob es ein Match ohne Zwischenstand (Score) gibt. Falls ja, wird eine getestet ob es sich im zuletzt gespielten Match ereignet hat. Falls ja übergibt dann die Methode alle Turns des aktuellen Spiels.

public int transformStringToInt(String number)

Diese Methode schreibt einen String in einen Integerwert um. Falls der String keinen Inhalt (null) hat oder leer ("") ist, wird Null zurückgegeben.

public String[][] catchWrongState()

Diese Methode testet ob es ein Match ohne Zwischenstand (Score) gibt. Falls ja, wird getestet, ob sich dies im zuletzt gespielten Match ereignet hat. Falls ja übergibt die Methode alle Turns des aktuellen Spiels.

deleteGame()

Diese Methode löscht in Abhängigkeit einer bestimmten Match- sowie Game-ID spezifisch ein komplettes Spiel.

7.5.2. DBConnector.java

Die DBConnector.java ist eine Schnittstellenklasse, welche teilweise auf die ausgelagerten Getter und Setter Methoden zugreift und teils direkt auf die ConnectHSQL. Weiterhin werden in dieser Methode die DB Threads gestartet und verwaltet. Die DBConnector trennt, die ConnectHSQL vom restlichen Programm ab, um Änderungen in der Datenbank auf einer anderen Flugebene ändern zu können.

7.5.3. Getter – Methoden .java

Im Package gibt es zahlreiche Getter Methoden, diese greifen auf die ConnectHSQL zu.



7.5.4. Setter – Methoden .java

Im Package gibt es zahlreiche Getter Methoden, diese greifen auf die ConnectHSQL zu. Die Setter- Methoden sind in Threads ausgelagert um neben der eigentlichen Programmalaufzeit abgearbeitet zu werden.

7.5.5. LegacyMethods.java

In LegacyMethods sind alte, nicht zu berücksichtigende Methoden abgespeichert.

7.5.6. Datenbank Aufbau

Zum weiteren Überblick über die Datenhaltung, die durch persistente Speicherung durch eine HSQL-Datenbank gewährleistet wird, ist nachfolgend ein ER-Diagramm eingefügt.

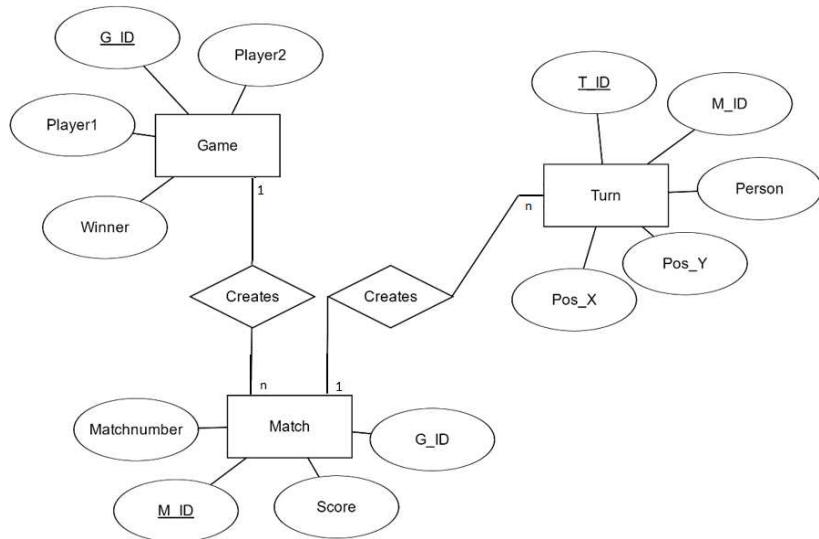


Abbildung 46: ER-Diagramm der Entitäten

Es wird zwischen drei Entitäten unterschieden. Dem Spiel (Game), dem Satz (Match) sowie dem Zug (Turn).

Der Entität Game werden vier Attribute zugeordnet, eine eindeutige Nummer, die G_Id, die hier als Primärschlüssel dient, Spieler 1 sowie Spieler 2, die in der GUI benannt werden können sowie der Gewinner des Spiels.

Der Entität Match werden drei Attribute zugeordnet, eine M_Id, die als fortlaufender, eindeutiger Beschreiber fungiert, eine Matchnumber, die sich zwischen den Werten 1-3 bewegt



und angibt, welcher Satz eines Spiels gerade gespielt wurde, die G_Id, die als Fremdschlüssel herangezogen wird sowie den Punktestand.

Der Entität Turn sind hingegen erneut vier Attribute zugeordnet, eine eindeutige Nummer, die T_Id, die als Primärschlüssel dient, eine M_Id, welche als Sekundärschlüssel fungiert sowie die Attribute Pos_Y, Pos_X und Person. Hierbei beschreibt das Attribut Person dabei, welcher Spieler den Zug durchgeführt hat, während Pos_Y die Zeile und Pos_X die Spalte angibt, in welche der Spielstein gesetzt wurde.

Die Beziehung zwischen Game und Match gestaltet sich so, dass einem Spiel n Züge zugeordnet werden. In diesem Fall ist n allerdings maximal drei, da nach drei Sätzen ein Spiel gewonnen wird.

Die Beziehung zwischen Match und Turn lässt sich ebenfalls durch eine 1:n-Beziehung beschreiben. Hierbei ist ein Zug einem Satz zugeordnet, während sich ein Satz aus mehreren Zügen zusammensetzt.

8. Programmablauf

Um den Programmfluss für Entwickler sowie Stakeholder nachvollziehbarer und transparenter zu gestalten, werden nachfolgend zwei Aktivitätendiagramme angehängt, die einen groben Überblick über den Ablauf eines Satzes sowie über den Aufruf bereits gespielter Spiele geben.

8.1. Satz spielen

Abbildung 47 zeigt hierbei das Aktivitätendiagramm für das automatische Spiel eines Satzes gegen die KI eines Gegners.

Es ist anzumerken, dass sowohl GUI, die Interfaces wie auch die Logik in der Bezeichnung „Agent“ zusammengefasst wurden.

Der Ablauf eines gespielten Satzes wird dargestellt und verdeutlicht somit die Interaktion der unterschiedlichen Komponenten untereinander.

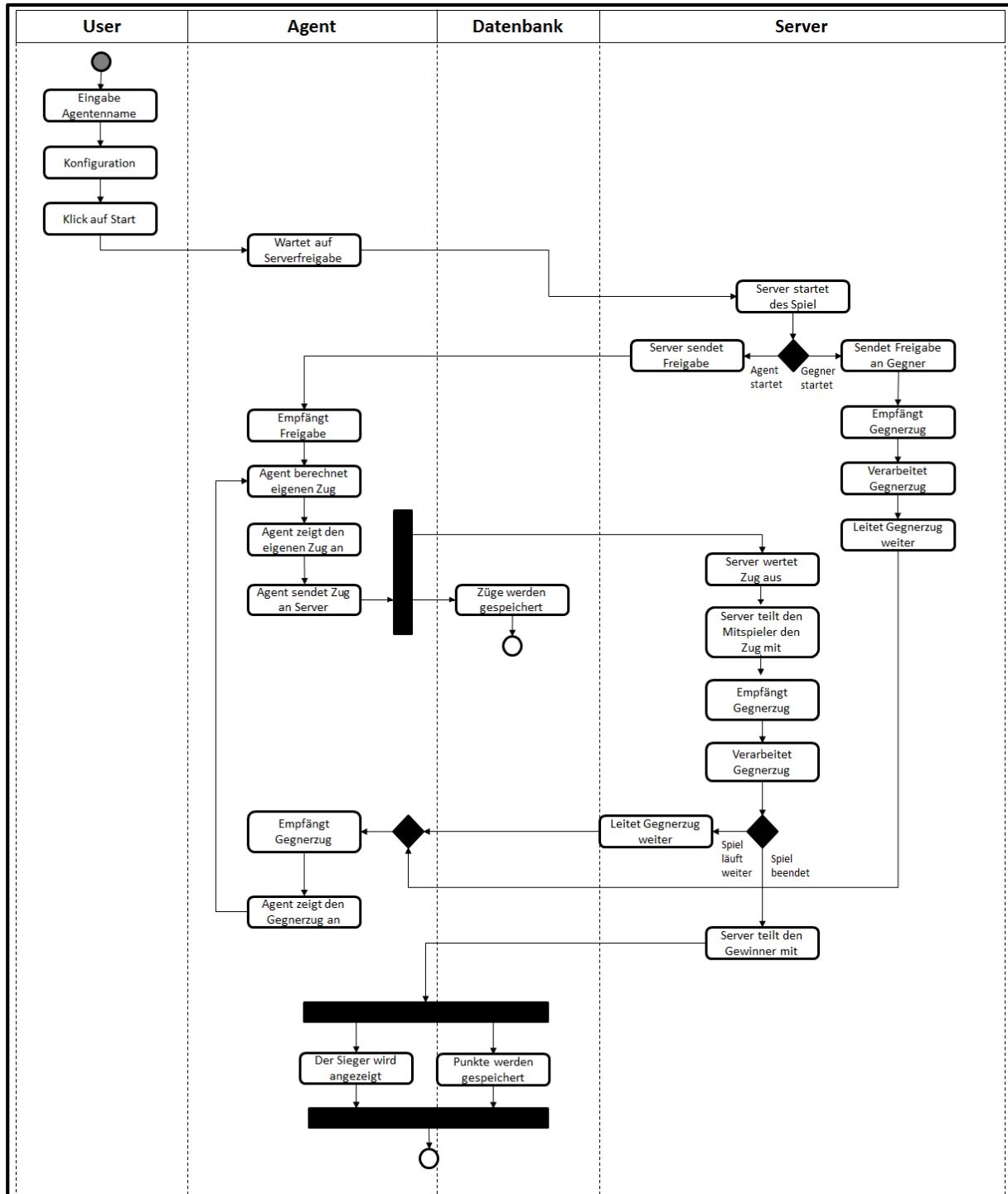


Abbildung 47: Aktivitätendiagramm "Satz spielen"



8.2. Gespielte Spiele nachverfolgen

Das zweite Aktivitätendiagramm bildet das Aufrufen und Nachverfolgen bereits gespielter Spiele ab. Hierbei kommt es zur Interaktion dreier Komponenten - des Users, der Datenbank sowie der GUI (s. Abbildung 48).

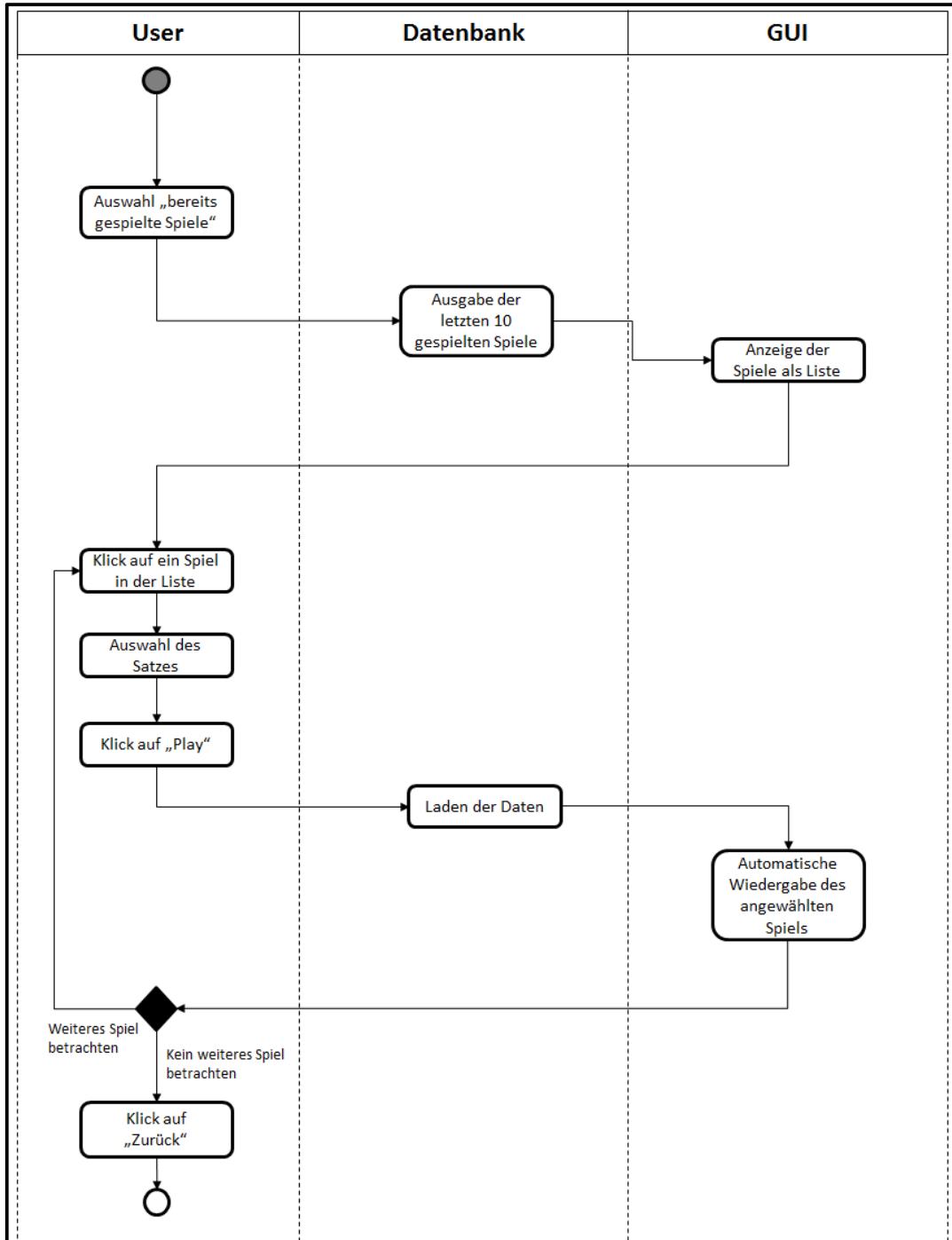


Abbildung 48: Aktivitätendiagramm "Spiel nachverfolgen"



9. JavaDoc

Die JavaDoc fasst die Dokumentationen aus Quelltexten im HTML-Format zusammen. Alle Klassen, Methoden sowie Interfaces sowie deren Interaktion können hierbei einfach und übersichtlich dargestellt werden.

Die JavaDoc von VIER. steht Ihnen entweder im ausgehändigten Projekt zur Verfügung oder kann direkt über folgenden Link angewählt werden.

<https://htmlpreview.github.io/?https://github.com/SvenC56/vierpunkt/blob/master/doc/index.html>

10. Referenz

Panitz, Sven Eric: Java will nur spielen : Programmieren lernen mit Spaß und Kreativität.
2. Aufl.. Berlin Heidelberg New York: Springer-Verlag, 2011.



Extras



The word "VIER" is written in large, bold, orange letters. The letters have a textured, hand-painted appearance with visible drips of orange paint hanging from the bottom, giving it a messy, artistic look.



(WWI14SCA, Praxisprojekt 2016)

Projekt: **VIER.**

Auftraggeber: **DHBW Mannheim**

Auftragnehmer: **WWI14SCA- Team VIER**

Leon Birk, Sven Cieslok, Tobias Jung, Jana Lang,
Angelina Neumann, Jana Schaub

Version	Datum	Autor	Kommentar
0.1	13.11.2016	A. Neumann	Neu angelegt



Extras

Die Applikation VIER. bietet Funktionalitäten, welche die funktionalen Anforderungen des Auftrags übertreffen.

- Dazu zählen unter anderem der thematische Wechsel der graphischen Oberfläche und somit die Möglichkeit, das Spielfeld nach den eigenen Vorstellungen anzupassen. Nicht nur die Hintergründe, sondern sämtliche Schaltflächen werden dabei ebenso farblich angepasst.
- Es ist möglich Spielernamen (Spieler, Gegner) zu verteilen und somit das Spiel zu personalisieren.
- Ferner ist es möglich gemeinsam manuell an einem PC gegeneinander zu spielen. Die Steine werden hierbei per Klick auf die Spielfläche gesetzt. So kann eine Partie VIER. nebeneinander gespielt werden.
- Die letzten zehn Spiele können nachverfolgt werden. Hierbei führt ein Klick auf „Play“ dazu, dass das Spiel automatisch rekonstruiert und jeder Zug nochmals angesehen werden kann.
- Das Spiel kann zudem jederzeit über den die Option im Menüreiter „Spiel verlassen“ beendet werden.