HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

INSTITUT
FÜR
SOFTWARE

Vier Gewinnt: Redesign Austellungsroboter-Software

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2012

Autor(en):	Colin Baumann, Dominique Wirz, Mike Schmid
Betreuer:	Eduard Glatz
Projektpartner:	Institut für Software, Rapperswil, M. Stolze
Experte:	-
Gegenleser:	-

Inhalt

1 Abstract 3
2 Management Summary 4
2.1 Ausgangslage 4
2.2 Vorgehen 4
2.3 Ergebnisse 4
2.4 Ausblick 5
3 Technischer Bericht der Arbeit 6
3.1 Analyse Problemstellung 6
3.1.1 Übersicht Aufbau 6
3.1.2 Domain Model 7
3.1.3 Use Cases 8
3.1.4 Analyse Varianten der Visualisierung 14
3.2 Design 19
3.2.1 Deployment Diagram 19
3.2.2 Logical View 20
3.2.3 Package Übersicht 21
3.2.4 Klassen Übersicht und Implementationsentscheide 23
3.2.5 Ablauf Spielerstellung und Spielzüge 27
3.2.6 Roboter Kommunikation 29
3.2.7 Serverseitige Implementation 30
3.2.8 Server – Client Kommunikation 31
3.2.9 Benutzeroberfläche und clientseitige Implementationen 34
3.2.10 Fehlerbehandlung 38
3.3 Algorithmen 39
3.3.1 WinChecker Algorithmus 39
3.3.2 GridEvaluator Algorithmus 40
3.3.3 TurnCalculator Algorithmus 41
3.3.4 Weitere Algorithmus Optimierungsmöglichkeiten 44
3.4 Testing 45
3.4.1 Analyse Tests 45
3.4.2 Unit Tests 46
3.4.3 System Tests 47
3.4.4 Usability Tests 48
3.5 Ergebnisse / Umsetzung 49
3.5.1 Erreicht 49

3.5.2 Nicht erreicht 49
3.6 Schlussfolgerungen 50
3.6.1 Stabilität 50
3.6.2 Wartung 50
3.6.3 Schwierigkeitsgrad 51
3.6.4 Visualisierung 51
3.6.5 Erweiterbarkeit & Austauschbarkeit 52
3.6.6 Gesamtfazit 53
4 Glossar 55
5 Literaturverzeichnis 56

1 Abstract

Der Vier-Gewinnt Roboter wurde als Semesterarbeit in einer interdisziplinären Arbeit von Studenten der Abteilung I und M entwickelt. Der Roboter dient als Demonstrationsobjekt an HSR-Werbe Events für zukünftige HSR-Studenten. Das Resultat der Arbeit aus dem Jahre 2010 entspricht einem Prototyp, der nicht alle optionalen Anforderungen zu erfüllen vermochte. Ebenfalls gestaltet sich eine Weiterentwicklung des Prototyps als schwierig.

Aufgabe dieser Arbeit war es den bestehenden Prototyp durch eine funktional ausgereiftere Software zu ersetzen. Ausserdem wird die Software so erweitert, dass dem interessierten Zuschauer eine Visualisierung des Spielalgorithmus dargeboten wird. Zusätzlich soll ein Demo-Modus realisiert werden, welcher bei längerer Nichtbenutzung durch eine Person ein automatisiertes Spiel startet.

Da der bestehende Code sehr unübersichtlich und fehlerhaft war, wurde die gesamte Software von Grund auf neu erstellt. In der überarbeiteten Version wurden ausserdem neue Konzepte wie beispielsweise HTML5-WebSockets eingeführt, sowie die gesamte Benutzeroberfläche vereinfacht und neu gestaltet. Die Idee des Spielalgorithmus wurde beibehalten (MiniMax + Alpha-Beta-Suche), jedoch wurde dieser neu implementiert und somit optimiert. Mittels Optimierungen wie z.B. Spielzug-Sortierung und intelligenten Bewertungsfunktionen konnte die Performance beträchtlich gesteigert werden. Die Einführung des Frameworks SpringSource hat es nebst einer sauberen Programmierung ermöglicht, die Nebenläufigkeiten zu eliminieren und einen stabilen Betrieb der Lösung zu garantieren. In dem zwei künstliche Intelligenzen automatisiert gegeneinander spielen, konnte der Demo-Modus erfolgreich realisiert werden. Durch die Montage eines Bildschirms am Roboter wird dem Besucher anhand der Visualisierung des Spielalgorithmus ermöglicht die Spiellogik des Computers live am Bildschirm zu verfolgen.

2 Management Summary

2.1 Ausgangslage

Im Rahmen einer interdisziplinären Semesterarbeit von Studenten der Abteilungen Informatik und Maschinenbau wurde ein Roboter entwickelt welcher fähig ist „Vier-Gewinnt“ zu spielen. Zu diesem Zweck wurde der Roboter mit einer künstlichen Intelligenz ausgestattet und dient heute der HSR als Demonstrationsobjekt an diversen Veranstaltungen (siehe Abb. 1).

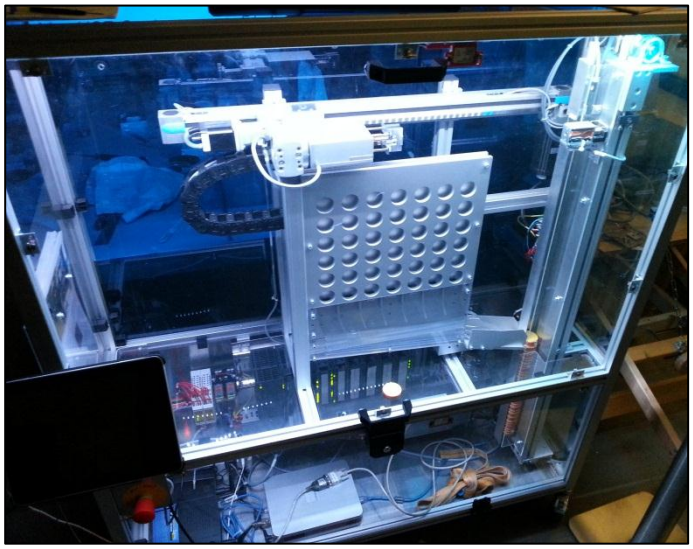


Abb. 1 Vier Gewinnt Roboter der HSR

Aufgrund der parallelen Entwicklung des Roboters und der dazu benötigten Software, hat sich der grösste Teil des Arbeitsaufwandes in der Spezifikation und Implementation der Schnittstellen niedergeschlagen. Ausserdem war der Roboter zur Zeit der Softwareentwicklung noch nicht vollständig Betriebsbereit, wodurch nicht alle Systemtests durchgeführt werden konnten.

Somit konnte der Roboter bestenfalls als Prototyp bezeichnet werden, welcher noch nicht alle gestellten Anforderungen zu erfüllen vermochte. Zudem war die Software schwer zu warten und auf Grund fehlerhafter Programmierung konnten nicht alle Problemstellungen abgefangen und gelöst werden.

2.2 Vorgehen

In einer ersten Phase wurde das bestehende Projekt analysiert und die bestehenden Probleme wurden mit dem Auftraggeber sowie den für die Wartung zuständigen Personen besprochen. In einer zweiten Phase wurde die gesamte Software des Roboters redesigned. In der dritten und letzten Phase wurde der Code implementiert und die Lösung getestet.

2.3 Ergebnisse

Der Code des bestehenden Projekts wurde als unwartbar bewertet. Es wurde ein neues Konzept für die Software erstellt. Erste Priorität war es mit dem neuen Design die Stabilität der Software für den Gebrauch des Roboters sicherzustellen. Zweite Priorität hatte die Wartbarkeit, so dass Fehler behoben oder weitere Projekte mit dem Roboter realisiert werden können. Durch einen Live-Test konnte die verbesserte Stabilität bestätigt werden. Kurz vor Abschluss des Projekts wurde die Software den Assistenten zum Reviewen gegeben und als gut befunden. Die Wartbarkeit sollte somit sichergestellt sein. Auf Wunsch des Auftraggebers wurde eine Visualisierung des Algorithmus des künstlichen Gegners realisiert.

2.4 Ausblick

Die neue Software des Vier Gewinnt Roboter ist nun auf einer soliden Basis, die einfach erweitert werden kann. Da der Fokus dieser Arbeit darauf lag, in erster Linie eine stabile Software zu produzieren, konnten nur wenige neue Features hinzugefügt werden. Folgende Punkte könnten verbessert werden:

- Die Visualisierung des Algorithmus könnte dem Zuschauer noch ausführlicher erläutert werden.
- Da der Standardisierungsprozess von HTML5 zum Zeitpunkt der Erarbeitung des Projekts noch nicht abgeschlossen war und die aktuelle Software-Lösung auf WebSockets aus HTML5 aufbaut, kann sich der Code mit der Weiterentwicklung von HTML5 ebenfalls verbessern.
- Die Rückmeldungen vom System gegenüber dem Benutzer, wie beispielsweise „der Roboter befindet sich gerade im Demo Modus“, könnten ausgebaut werden.

3 Technischer Bericht der Arbeit

3.1 Analyse Problemstellung

3.1.1 Übersicht Aufbau

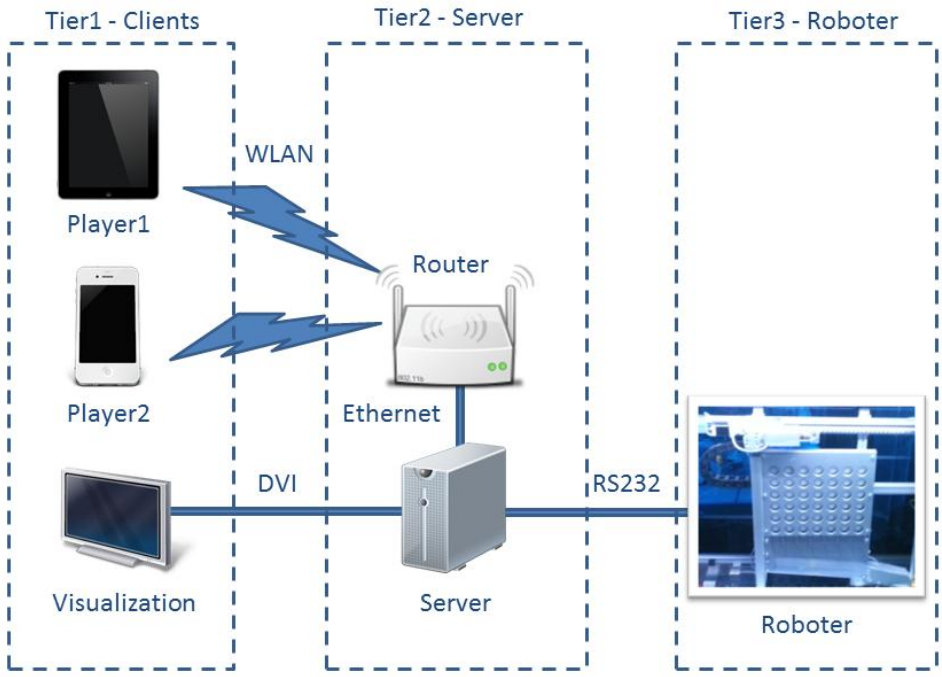


Abb. 2 Übersicht der Hardware

Die verschiedenen Geräte lassen sich grob in drei Tiers einordnen(siehe Abb. 2):

Auf der Benutzerebene (Tier1 – Clients) kann sich ein Besucher mittels eines WIFI-fähigen Gerätes auf den eigens für den Roboter konfigurierten WLAN-Router verbinden. Mittels eines modernen Webbrowsers kann über die Benutzeroberfläche anschliessend eine „Vier Gewinnt"-Partie gestartet werden. Sollte der Besucher nicht über ein oben definiertes Gerät verfügen, stehen diesem beim Roboter zwei iPads für die Benutzung zur Verfügung.

Des Weiteren befindet sich auf dieser Ebene ein weiterer Client in Form eines am Roboter angebrachten Bildschirmes, auf welchem der Spielalgorithmus visualisiert wird. Die Visualisierung funktioniert ebenfalls via Browser und dient nur zur Anzeige. Vom Benutzer ist hier keine Interaktion notwendig.

Die zweite Ebene (Tier2 – Server) stellt die Schnittstelle zwischen dem Benutzer und dem Roboter in Form eines Servers dar. Auf dem Server befindet sich die Ablauf- und Spiellogik, wobei die Koordination der Kommunikation zwischen den Clients und dem Roboter als auch die Steuerung der künstlichen Intelligenz zu dessen Aufgaben gehört.

Via RS232 Verbindung werden die definierten Befehle an den Roter gesendet. Ebene drei (Tier3 – Roboter) stellt die notwendige Roboterhardware zur Verfügung. Der Roboter arbeitet die eingehenden Befehle sequenziell ab. Da der Roboter über keinerlei Spielintelligenz verfügt, wird die Anzahl sowie die Anordnung der eingeworfenen Steine im Server gespeichert und stetig nachgeführt.

3.1.2 Domain Model

Das Domain Model dient zur übersichtlichen Darstellung der Problemdomäne. Da es sich dabei um eine Abstraktion der Realität handelt, wurde es während der gesamten Entwicklungsphase nicht mehr verändert. Schliesslich ist die konkrete Implementation der Klassen in einem Domain Model nicht ersichtlich. Auf der Grundlage des Domain Models wurden später die Klassen Diagramme erstellt (siehe Kapitel 4.2.4).

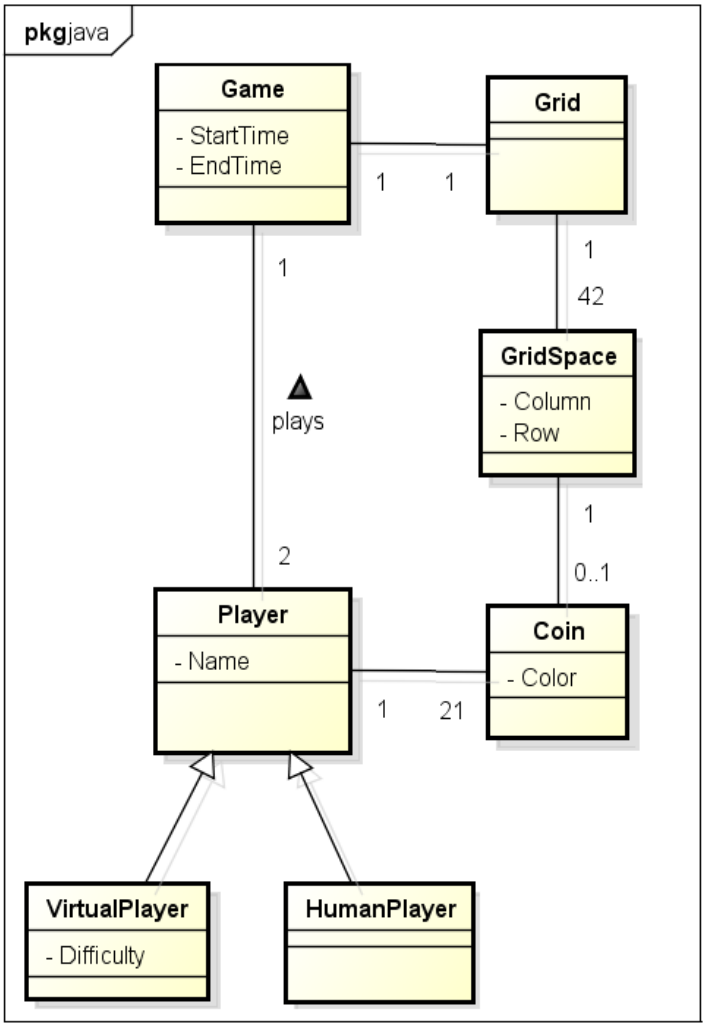


Abb. 3 Übersicht der Problemdomäne

Die Problemdomäne des Viergewinnt Spiels ist simpel aufgebaut. Zwei Spieler *(Player),* seien sie nun menschlich*(HumanPlayer)* oder von einer künslichen Intelligenz *(VirtualPlayer)*, spielen gegeneinander ein Spiel *(Game)*. Dabei besitzt jeder Spieler 21 Spielsteine *(Coin)* mit seiner jeweiligen Farbe. Jedes Spiel wird auf einem Spielfeld *(Grid)* ausgetragen, in welches abwechselnd Spielsteine eingeworfen werden. Das Spielfeld ist jeweils aufrecht stehend und besitzt 42 Löcher *(Gridspaces)* welche in einem Rechteck (7x6) angeordnet sind. Wird nun ein Spielstein eingeworfen, so fällt dieser bis zum untersten freien Loch durch. Somit kann jedes Loch entweder einen oder keinen Spielstein enthalten.

In diesem Modell wurden explizit Roboter oder Clients weggelassen, da diese nicht zur Problemdomäne gehören, sondern nur Visualisierungen des Spielfeldes darstellen.

3.1.3 Use Cases

3.1.3.1 *Use Case Diagram*

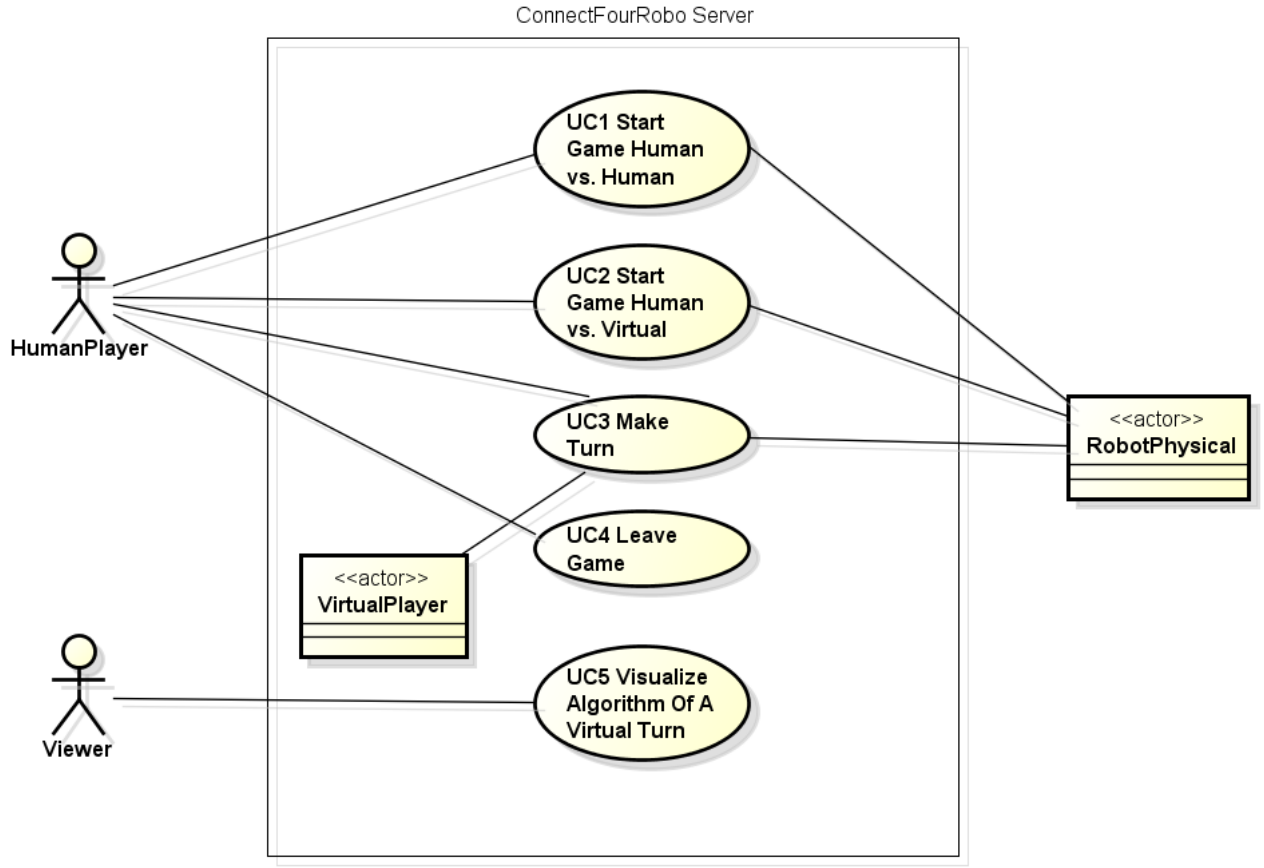


Abb. 4 Auflistung der wichtigsten Use Cases

Human Player und Viewer sind beide menschliche Akteure ausserhalb des Systems. Wobei der Humanplayer aktiv das System beeinflussen und der Viewer lediglich das System Anfragen kann und danach Visualisierungsdaten erhält.

Der Virtual Player ist kein Mensch und befindet sich innerhalb des Systems, beeinflusst das System mit seinen Handlungen aber aktiv.

Der physische Roboter ist kein Primärakteur, er reagiert nur auf Anfragen bzw. Befehle des Systems. Ausserdem ist der physische Roboter unabhängig von diesem System und deshalb ausserhalb der Systemgrenze.

3.1.3.2 *UC1 Start Game Human vs. Human*

Primary Actor: Human Player (nachfolgend menschlicher Spieler genannt)

Overview: Der menschliche Spieler gibt einen Namen an und entscheidet sich für ein Spiel gegen einen menschlichen Gegenspieler.

Stakeholder and Interests:
- Menschlicher Spieler: Möglichst mit wenig Aufwand ein Spiel spielen.
- HSR: Spieler für den Vier Gewinnt Roboter begeistern.

Preconditions:
- Roboter ist betriebsbereit
- Ein zweiter menschlicher Spieler wartet bereits im Spiel oder es wird in geraumer Zeit ein menschlicher Spieler dazu stossen.

Postconditions:
- Der menschliche Spieler befindet sich in einem neuen Spiel gegen einen menschlichen Spieler.

Main Success Scenario:
1. Menschlicher Spieler erscheint vor dem Roboter und möchte ein Spiel gegen einen Menschen spielen.
2. Menschlicher Spieler befolgt die Anleitung am Roboter und begibt sich auf die Lobby-Webseite.
3. Menschlicher Spieler gibt seinen Namen ein und startet das Spiel.
4. Menschlicher Spieler muss warten bis ein zweiter menschlicher Spieler dazu kommt.
5. System gibt dem Roboter den Befehl das alte Spiel im Roboter abzuräumen
6. Menschlicher Spieler befindet sich mit einem anderen menschlichen Spieler in einem neuen Spiel.

Extensions (or Alternative Flows):

4a. Es war bereits ein anderer menschlicher Spieler am Warten.
   1. Spieler muss nicht auf zweiten menschlichen Spieler warten.

3.1.3.3 *UC2 Start Game Human vs. Virtual*

Primary Actor: Human Player (nachfolgend menschlicher Spieler genannt)

Overview: Der menschliche Spieler gibt einen Namen an und entscheidet sich für ein Spiel gegen einen Virtual Player (nachfolgend künstlicher Gegenspieler genannt). Beim Erstellen des Spiels gibt der menschliche Spieler die gewünschte Stärke des künstlichen Gegenspielers an.

Stakeholder and Interests:
- Menschlicher Spieler: Auch ohne menschlichen Gegenspieler ein Spiel spielen.
- Menschlicher Spieler: Auch mit künstlichem Gegenspieler ein angemessenen Gegner haben (nicht zu schwer, nicht zu leicht)
- HSR: Spieler für den Vier Gewinnt Roboter begeistern.

Preconditions:
- Roboter ist betriebsbereit
- Es ist noch kein menschlicher Spieler in einem Human vs. Human Spiel am Warten.

Postconditions:
- Der menschliche Spieler befindet sich in einem neuen Spiel gegen einen virtuellen Spieler.

Main Success Scenario:
1. Menschlicher Spieler erscheint vor dem Roboter und möchte ein Spiel gegen einen künstlichen Gegenspieler spielen.
2. Menschlicher Spieler befolgt die Anleitung am Roboter und begibt sich auf die Lobby-Webseite.
3. Menschlicher Spieler gibt seinen Namen ein, wählt die Schwierigkeitsstufe des künstlichen Gegenspielers und startet das Spiel.
4. System gibt dem Roboter den Befehl das alte Spiel im Roboter abzuräumen.
5. Menschlicher Spieler befindet sich mit einem künstlichen Gegenspieler in einem neuen Spiel.

3.1.3.4 *UC3 Make Turn*

Primary Actor: Player (Dies kann ein Virtual- oder Human Player sein, nachfolgend Spieler genannt)

Overview: Der Spieler befindet sich in einem Spiel und möchte einen Turn (nachfolgend Zug genannt) machen.

Stakeholder and Interests:
- Spieler: Möglichst intuitiv und simpel wie in einem echten Vier Gewinnt Spiel einen Zug machen.
- HSR: Spieler generell für Informatik begeistern.

Preconditions:
- Roboter ist betriebsbereit
- Es wurde ein Human vs. Human Spiel zuvor erstellt, oder ein Human vs. Virtual Spiel. Siehe UC1 und UC2.
- Der Spieler ist am Zug.
- Das Spiel ist noch nicht zu Ende.

Postconditions:
- Der Spielerzug wurde erfolgreich durchgeführt im sowohl am Roboter als auch auf dem Gerät des Spielers.
- Der Gegenspieler ist am Zug.

Main Success Scenario:
1. Spieler klickt in eine Spalte um einen Zug zu machen.
2. System führt Zug auf dem Roboter aus und simuliert Zug an den Geräten von Spieler1 und Spieler2.
3. System setzt den Gegenspieler als nächsten Spieler der am Zug ist.
4. System überprüft ob das Spiel zu Ende ist durch einen Gewinner oder durch Unentschieden.

Extensions (or Alternative Flows):

4a. Das Spiel ist nicht zu Ende.
1. Gegnerischer Spieler ist wieder am Zug (UC3 beginnt von vorne).

4b. Das Spiel ist zu Ende mit einem Sieger.
1. System sendet dem Roboter den Befehl mit der Siegerfarbe die Siegerpositionen anzuzeigen.
2. System Zeigt Sieger und Verlierer auf dem jeweiligen Gerät an.
3. System beendet das Spiel.
4. System leitet Spieler wieder in die Spiellobby.

4c. Das Spiel ist zu Ende mit einem Unentschieden
1. System zeigt das Unentschieden auf beiden Spielergeräten an.
2. System beendet das Spiel.
3. System leitet Spieler wieder in die Spiellobby.

*a. Jederzeit wenn ein Spieler das Spiel verlassen möchte.
1. Fortsetzung in UC 4

3.1.3.5 *UC4 Leave Game*

Primary Actor: Human Player (nachfolgend menschlicher Spieler genannt)

Overview: Der menschlicher Spieler befindet sich in einem Spiel und verlässt das Spiel bevor es einen Sieger oder ein Unentschieden gegeben hat.

Stakeholder and Interests:
- menschlicher Spieler: Da er die Absicht hat nun etwas anderes zu machen, möchte er das Spiel mit möglichst wenig Ablenkung bzw. Aufwand verlassen.
- HSR: Möchte guten Eindruck beim menschlicher Spieler hinterlassen.

Preconditions:
- Der menschliche Spieler befindet sich in einem Spiel.
- Das Spiel ist noch nicht zu Ende.

Postconditions:
- Der menschliche Spieler befindet sich in keinem Spiel (Spiel wurde korrekt beendet).

Main Success Scenario:
1. menschlicher Spieler klickt auf den Button Spiel verlassen.
2. System beendet Spiel.
3. System leitet den übrig gebliebenen menschlicher Spieler in die Spiellobby.

Extensions (or Alternative Flows):
1a. Der Spieler klickt auf den Button Browser schliessen.
3a. Der Gegenspieler ist ein Virtual Player. System räumt Virtual Player ab.

3.1.3.6 *UC5 Visualize Algorithm of A Virtual Turn*

Primary Actor: Viewer

Overview: Es läuft ein Spiel mit einem menschlichen und einem künstlichen Spieler. Zusätzlich zeigt nun das Visualisierungs GUI Grafiken an welche die Zugfindung des künstlichen Spielers aufzeigt.

Stakeholder and Interests:
- Viewer: Möchte Einblicke in die Zugfindung des künstlichen Spielers.
- HSR: Möchte die Attraktivität des Roboters steigern mit einer Visualisierung.
- HSR: Möchte den Informatik Teil des gesamten Projekts in den Vordergrund rücken.
- HSR: Möchte das Interesse für Informatik wecken um Leute für den IT-Studiengang zu begeistern.

Preconditions:
- Es besteht ein Human vs. Virtual Game.
- Das Spiel ist noch nicht zu Ende.
- Der künstliche Spieler hat seinen Zug bereits gemacht, der Zug wurde aber noch nicht an den Roboter gesendet.
- Das Visualisierungs GUI hat sich am System angemeldet um Daten zur Visualisierung zu erhalten.

Postconditions:
- Der menschliche Spieler ist am Zug.

Main Success Scenario:
1. System stellt die Visualisierung des Algorithmus dar.
2. Viewer sieht sich die Visualisierung an.
3. Wird ein neuer virtueller Zug gemacht, wird die Visualisierung aktualisiert.

Schritte 1-3 werden nach jedem Zug (UC3) des künstlichen Spielers wiederholt.

3.1.4 Analyse Varianten der Visualisierung

Um den Spielalgorithmus für den Zuschauer interessant zu gestalten, wurden verschiedene Möglichkeiten in Betracht gezogen. Alle Vorschläge sowie unsere Empfehlungen wurden dem Auftraggeber vorgelegt, welcher sich dann für eine Variante entscheiden musste.

3.1.4.1 *Visualisierung durch Balken + Logger:*

Der Algorithmus könnte als eine Hybridform dargestellt werden, welche einerseits die „Güte“ eines Zuges als auch eine Log-Artige Ausgabe der Berechnungen ausgibt (siehe Abb. 5).

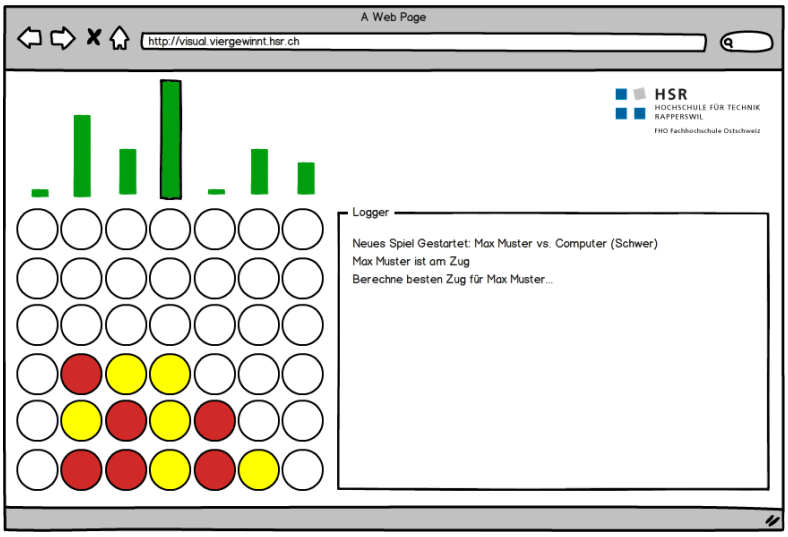


Abb. 5 Visualisierung in Form von Balken und Logger

Die grünen Balken über den Spalten zeigen an, wie vorteilhaft es für den aktuellen Spieler sein würde, einen Stein in die entsprechende Spalte zu werfen. Je höher der Balken desto besser. Diese Balken könnten sich mittels einer Animation während der Berechnung laufend ändern.

Rechts wird ein Logger angezeigt, welcher textuell beschreibt, was der Roboter gerade „Denkt“. Hier könnte die Ausgabe beliebig detailliert erfolgen. Entweder lediglich die Findung des besten Zuges, oder aber auch Ausgaben, wie z.B. „warte auf Spieler xy“, „werfe Stein in Spalte“ etc.

Pro / Kontra

Grösster Vorteil dieser Lösung ist die einfache Implementation. Es müssen jeweils nur die Werte der obersten Stufe des Algorithmus ausgewertet und der Logger auf die Visualisierung umgeleitet werden. Ausserdem könnte der Logger zu Entertainment-Zwecken aufgepeppt werden. z.B. wenn ein zufälliger Zug gemacht wird, könnte eine Ausgabe wie: „Ich will dich auch mal gewinnen lassen…“ gemacht werden. Dies wäre unterhaltsamer für etwaige Zuschauer.

Allerdings wird bei dieser Variante nicht die eigentliche Zugfindung visualisiert, sondern das Ergebnis daraus. Der Betrachter kann daraus nicht lesen, welche Berechnungsmethode gewählt wurde. Ausserdem wird der Informatik-Teil des Roboters zu wenig hervorgehoben.

Fazit

Dies ist die einfachste aller Möglichkeiten der Visualisierung. Allerdings ist sie auch recht rudimentär und erfüllt ihren eigentlichen Zweck nicht vollständig: die Visualisierung des Algorithmus.

3.1.4.2 *Visualisierung durch Baumstruktur*

Der Algorithmus könnte in Form einer Baumstruktur visualisiert werden (siehe Abb. 6). Dabei wird der Algorithmus eins zu eins abgebildet.

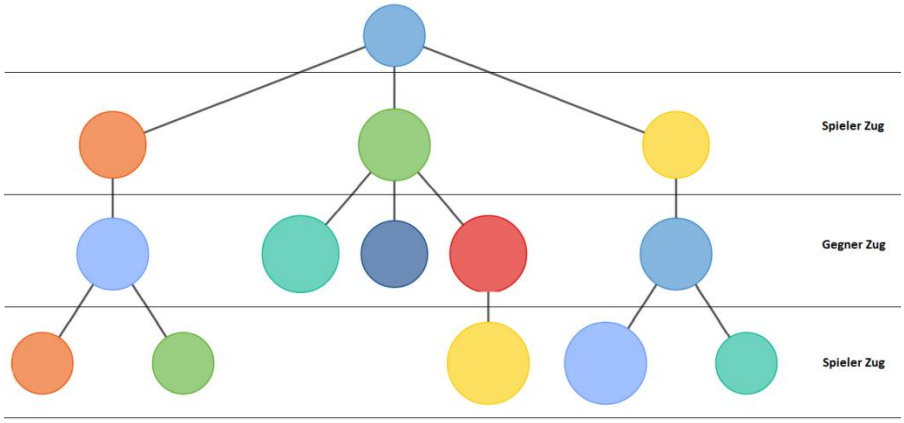


Abb. 6 Visualisierung in Form einer Baumstruktur

Der Baum zeigt alle möglichen Züge der beiden Spieler auf. Mit fortschreitender Suchtiefe wechseln sich die möglichen Züge des aktuellen Spielers mit den möglichen Zügen des Gegners ab. Die Blattkonten enthalten jeweils die Bewertungen der verschiedenen Spielsituationen welche dann weiter in Richtung Root-Knoten propagiert werden.

Da sich der Baum mit exponentiellem Wachstum vergrössert ($7^{suchTiefe}$), würde er schnell die Darstellungsmöglichkeiten jedes Bildschirmes übersteigen. Um dieser Problematik entgegen zu wirken gibt es zwei Möglichkeiten:

1. Die dargestellte Suchtiefe begrenzen
2. Nur den „Pfad“ des besten Zuges genauer anzeigen.

Pro / Kontra

Da der Algorithmus sowieso schon einen Baum erstellt, wäre es relativ einfach diesen noch in einer reduzierten Variante zu visualisieren. Entsprechende JavaScript -Bibliotheken sind bereits verfügbar. Ausserdem sehen die Beobachter des Roboters auf eindrückliche Weise, wie der Algorithmus im Hintergrund arbeitet.

Das Problem mit dem begrenzten Platz auf einem Bildschirm ist nur schwer in den Griff zu kriegen. Selbst bei einer Suchtiefe von 3, wären auf der untersten Ebene immer noch 7^3=343 Knoten vorhanden. Bei einer theoretischen Bildschirmbreite von 1920 Pixeln würden nur noch ca. 5 Pixel pro Knoten verfügbar sein. Dabei hätten die Knoten keinen Abstand zwischen einander und auch das enthalten von Text in den Knoten wird verunmöglicht. Wenn nur der beste Pfad besser visualisiert wird, ist die Aufbereitung der Daten aus dem Algorithmus weitaus komplexer und nur mit einem erheblichen Mehraufwand verbunden.

Fazit

Knackpunkt bei dieser Variante der Visualisierung ist zweifelsohne der begrenzte Platz. Ausserdem hängt die saubere Darstellung auch von den Möglichkeiten der verwendeten JavaScript Bibliothek ab.

Falls diese Version gewählt werden soll, kann auf jeden Fall nicht der gesamte Baum visualisiert werden, sondern nur ein Teil davon.

3.1.4.3 *Visualisierung durch Code Highlighting*

Eine weitere Visualisierungsmöglichkeit wäre, dem interessierten Besucher den Algorithmus Code in Form einer Debugging-Konsole zu präsentieren, in welcher die einzelnen Code Zeilen durchlaufen werden (siehe Abb. 7).

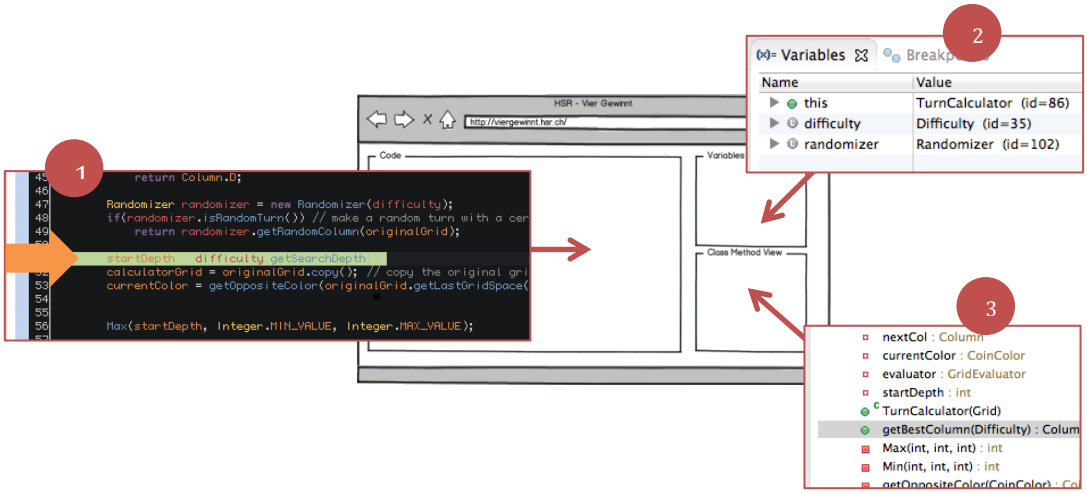


Abb. 7 Visualisierung mittels Code-Highlighting

In dieser Visualisierung würde der Fokus auf das Durchlaufen des Codes gelegt werden. Es werden drei Hauptbereiche definiert:

1. **Code:** Der Code wäre sichtbar und ein Cursor markiert die aktive Zeile.
2. **Variablen:** Hier würden die verwendeten Variablen dargestellt werden und je nach Methode angepasst werden.
3. **Klassenmethoden:** Dieser Bereich würde verwendet werden um die aktuelle Methode im Code zu zeigen.

Der zu durchlaufende Code würde sich hierbei auf die Berechnung des nächsten Zugs beziehen. Das Durchlaufen wäre jedoch nur eine Animation auf dem Browser, wobei jeweils beim Start der eigentlichen Berechnung im Code die Animation gestartet wird und nach Erhalt des Computer berechneten Zugs beendet wird.

Pro / Kontra

Für diese Variante spricht sicherlich, dass der Besucher einen realitätsnahen Einblick in das Tun eines Programmierers erhält. Hierdurch könnte einem technisch interessierten Besucher der Studiengang Informatik schmackhaft gemacht werden. Hinzu kommt, dass mit der Animation im Browserfenster das Interesse geweckt wird.

Da der Code normalerweise sehr schnell durchlaufen wird, müsste das Abarbeiten des Codes zur Animation mit einer Zeitverzögerung verlangsamt werden. Jedoch würde der Spielfluss somit erheblich beeinträchtigt werden, was gegen die Umsetzung dieser Variante spricht. Ein weiterer negativer Punkt ist, dass es sich hierbei um eine rein textuelle Darstellung handelt, was bei einem schnellen Blick nicht interessant wirken kann.

Fazit

Diese Lösung bietet dem Besucher einen tieferen Eindruck in den Bereich Informatik. Leider wäre die Animation nicht mit dem Ablauf des Programms synchronisiert, wodurch eine ver-

fälschte Ansicht vermittelt werden könnte. Des Weiteren ist diese Visualisierung im Vergleich zu einer Bildlichen Darstellung des berechneten Baums nicht intuitiv.

3.1.4.4 *Visualisierung durch „Path"-Verfolgung*

In dieser Lösung geht es darum den Pfad, welcher vom MiniMax-Algorithmus ausgewählt wurde, möglichst sinnvoll darzustellen (siehe Abb. 8).

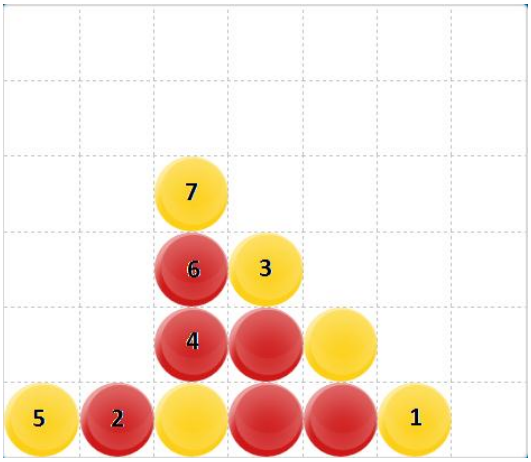


Abb. 8 Visualisierung der Reihenfolge der Coins (Nicht nummerierte Coins sind Coins die bereits gelegt wurden.)

Da es bereits bei geringer Suchtiefe nicht mehr möglich ist den gesamten Baum des MiniMax-Algorithmus auf einem Bildschirm darzustellen, da dieser extrem schlecht skaliert, wäre es naheliegend lediglich den relevanten Ast des Baumes anzuzeigen. Da dies dann aber nur noch ein Pfad mit n Knoten ist, wobei n die Anzahl voraus berechneter Züge wäre und jeder Knoten einen Zug repräsentiert, wäre eine Darstellung in Baumform eher uninteressant.

Da dem Spieler bei dieser Methode eine Liste der nächsten Züge vorliegt, welche vom MiniMax-Algorithmus als beste Option berechnet wurden, bietet sich bei dieser Variante das dem Spieler bereits vertraute Vier-Gewinnt-Grid als intuitivste Darstellung an. Somit wäre in diesem Grid der optimale Ablauf des Spiels aus Sicht des aktuellen Spielers ersichtlich, wobei jeder gespielte Coin mit einer aufsteigenden Nummer versehen wäre.

Pro / Kontra

Ein Vorteil dieser Methode ist, dass diese Darstellungsform ohne Kenntnisse von mathematischen Bäumen verstanden werden kann. Da sich der Algorithmus bei gleichem Spielverhalten nicht ändert, kann sich der Spieler durch das Merken der zu legenden Steine gute Spielstrategien zusammenstellen und so sein Wissen erweitern.

Andererseits hat dieses Konzept auch einige Nachteile. Spieler könnten somit optimal spielen, sofern Sie sehen was der Algorithmus in ihrer Situation tun würde. Oder der optimale Zug für den Virtual Player könnten vor zu verhindert werden, was dann vermutlich immer auf ein Unentschieden hinauslaufen würde.

Ausserdem würde diese Darstellung keine grosse Attraktivitätssteigerung darstellen, da lediglich ein weiteres Spielfeld gezeichnet wird. Hierbei würden Darstellungsmethoden welche komplett anders sind als das Grid vermutlich mehr Personen dazu bringen sich den Roboter anzuschauen.

Fazit

Mit diesem Ansatz wird die Problematik, dass viele Benutzer mit Baumstrukturen nicht vertraut sind, einfach gelöst. Jedoch trägt dies nur wenig zur Einsicht in die Funktionsweise des Algorithmus bei. Ausserdem geht die Chance verloren weitere Personen auf den Roboter aufmerksam zu machen, da diese Lösung exakt gleich aussieht wie das bereits bestehende Spielfeld.

3.1.4.5 *Entscheidung*

Nach längerer Analyse wurde ermittelt, dass sich eine abgewandelte Baumvisualisierung des Algorithmus gesamthaft betrachtet, als optimalste Lösung empfiehlt. Sie ist relativ einfach zu implementieren und stellt den wirklichen Kern des Algorithmus dar (MiniMax-Baum). Ausserdem hat der Kunde den Wunsch geäussert die Visualisierung anhand mathematischer Bäume zu gestalten, wobei dies durch das Projektteam eingehalten werden möchte. Dazu wurde seitens des Auftraggebers eine gute Library zur Visualisierung von Bäumen vorgeschlagen.

3.2 Design

3.2.1 Deployment Diagram

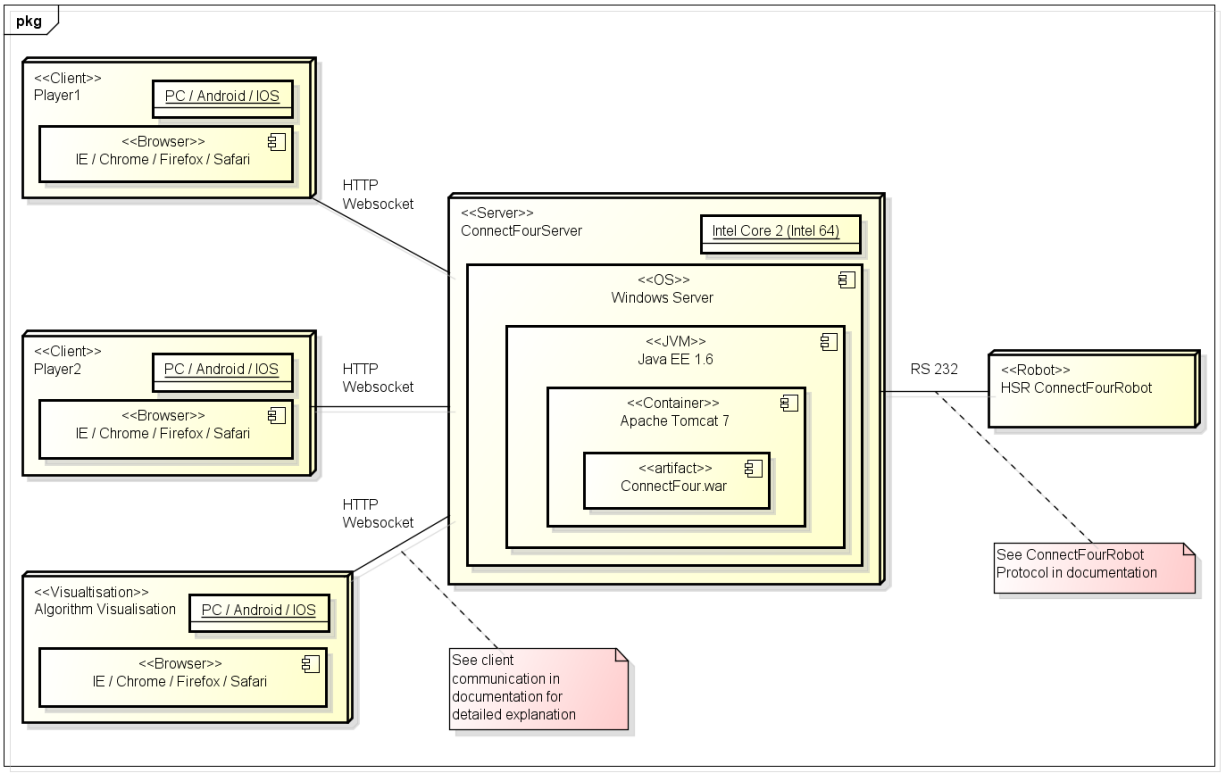


Abb. 9 Darstellung der Deployment Umgebung

Clients können mit einem Computer, Tablet oder Smartphone, auf welchen Internet Explorer, Chrome, Firefox oder Safari installiert ist, über HTTP WebSockets mit dem Server kommunizieren. Für die Visualisierung gilt das gleiche, wobei diese in der momentanen Lösung im Browser des Servers geöffnet wird.

Zur Zeit wird ein Windows Server eingesetzt. Da die Software mit Java implementiert wurde, könnten auch andere Betriebssysteme eingesetzt werden. Als Webserver und somit auch Container wurde Tomcat 7 verwendet.

Die Kommunikation mit dem Vier Gewinnt Roboter (ConnectFourRobot) erfolgt per RS 232. Da die Entwicklung und Umsetzung des physischen Roboters nicht Bestandteil dieses Projekts ist, wird dessen Aufbau nicht näher erläutert.

3.2.2 Logical View

Die Packages bauen wie folgt aufeinander auf:

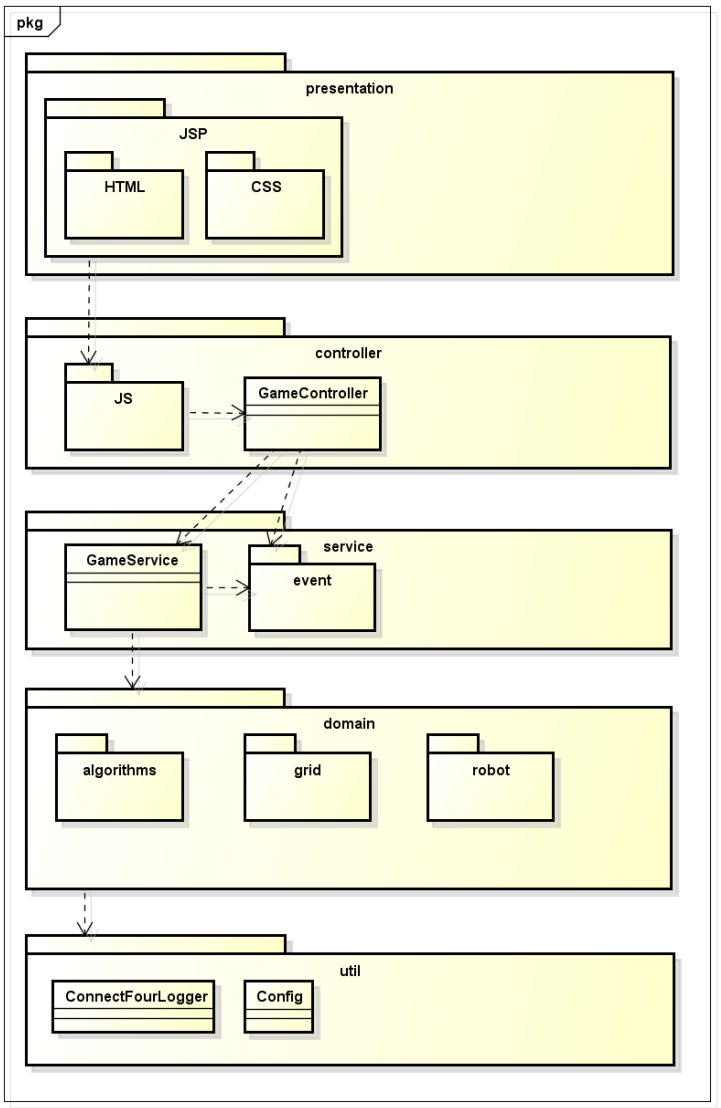


Abb. 10 Übersicht der verschiedenen Layer

Auf der Präsentationsschicht wurden, wie in Abbildung 10 ersichtlich, die Technologien HTML und CSS eingesetzt.

Das JavaScript Package wird in der Controller Schicht geführt, weil es auf der Client Seite für die Interaktion mit der Benutzeroberfläche zuständig ist und die Kommunikation mit dem Controller löst.

Der darunterliegende Service Layer ist für die eigentliche Instanziierung und Verwaltung der Domain-Objekte verantwortlich.

Die Domain Schicht beinhaltet sowohl alle Domainelemente wie z.B. Grid und Roboter, als auch die gesamte Spiellogik. Diese Domain Schicht könnte auch unabhängig von ihrer Umgebung in einem anderen Projekt verwendet werden.

Auf der untersten Ebene befindet sich die Util Schicht, welche allgemeine Funktionalitäten wie z.B. den Zugriff auf die Konfiguration oder das Log bereitstellt.

3.2.3 Package Übersicht

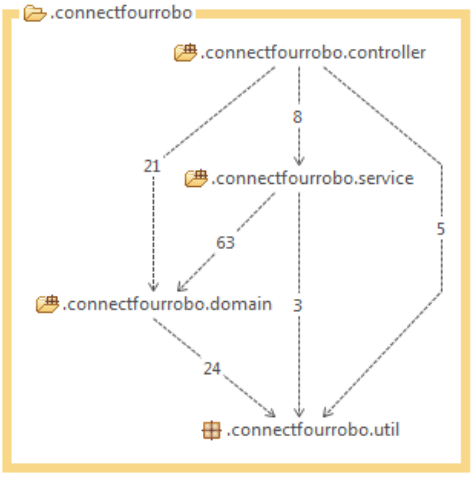


Abb. 11 Übersicht der Main-Packages

Das gesamte Projekt wurde grob in vier Packages gegliedert. (*Controller, Service, Domain, Util*). Im folgenden Kapitel wird auf die Struktur und den Zweck der einzelnen Packages eingegangen. Die hier verwendeten Grafiken wurden alle dem Struktur-Analyseprogramm STAN entnommen, welches als Freeware Eclipse Plugin verfügbar ist. Zu sehen sind jeweils die Packagenamen, sowie die Anzahl und Richtung der Abhängigkeiten zu einander. In der Abbildung 11 ist beispielsweise zu erkennen, dass es aus dem Package *Service* drei Abhängigkeiten zu *Util* gibt, allerdings keine von *Util* zurück zu *Service.*

Package „Controller“:

Beim *Controller* Package handelt es sich um ein Package welches die Schnittstelle zu den Clients darstellt. Es enthält hauptsächlich die Klasse *GameController*, welche zur Aufgabe hat, Client-Requests an die entsprechende Service Klasse weiterzuleiten.

Das SubPackage *Support* enthält Helferklassen um unter anderem eingehende Standard JAVA HTTP-Requests in AtmosphereResources einzupacken. Alle in diesem Package enthalten Klassen wurden aus dem Atmosphere Beispiel übernommen.

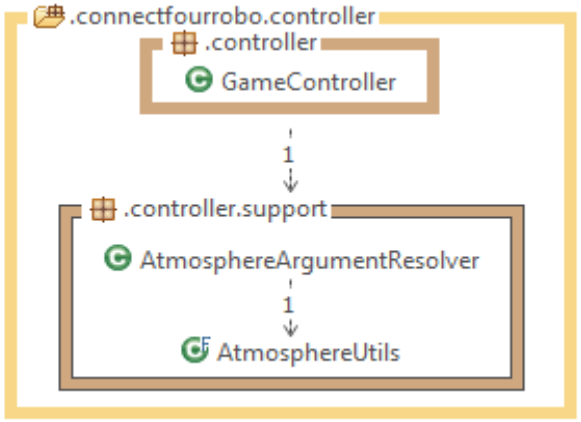


Abb. 12 Controller Package im Detail

Package „Service“:

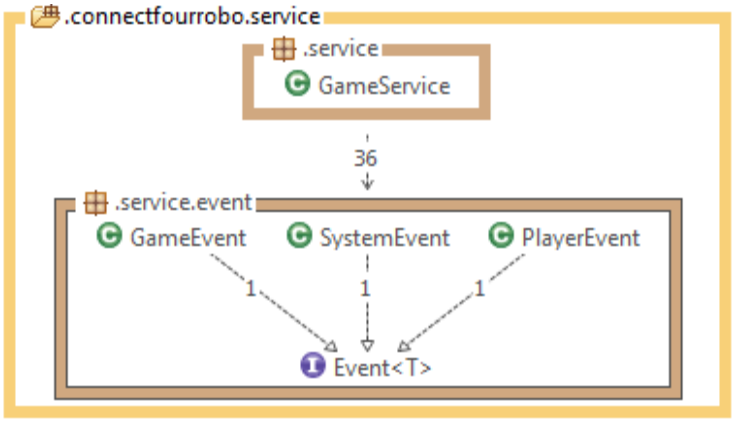


Abb. 13 Service Package und Events

Im Package *Service* sind die Klassen untergebracht welche zur Verarbeitung der Clientanfragen wichtig sind. Die *GameService* Klasse verwaltet das Spiel und die dazugehörigen Spieler. Des Weiteren verwendet diese Klasse das Subpackage *event,* wobei Instanzen der enthaltenen Klassen zwischen Server und Client ausgetauscht werden.

Package „Domain“:

Das Package *Domain* ist bezüglich Umfang weitaus das Grösste. Aus diesem Grund wurde dies wie folgt unterteilt.

Im Subpackage *domain.robot* befinden sich alle Roboterklassen, sowie die Klassen, welche für die serielle Kommunikation zum physikalischen Roboter abwickeln.

Die künstliche Intelligenz, welche für das Spiel gegen einen Computergegner benötigt wird, befindet sich im Subpackage *domain.turncalculator*

Alle Domainklassen verwenden das *Grid* und greifen somit auf dieses zu.

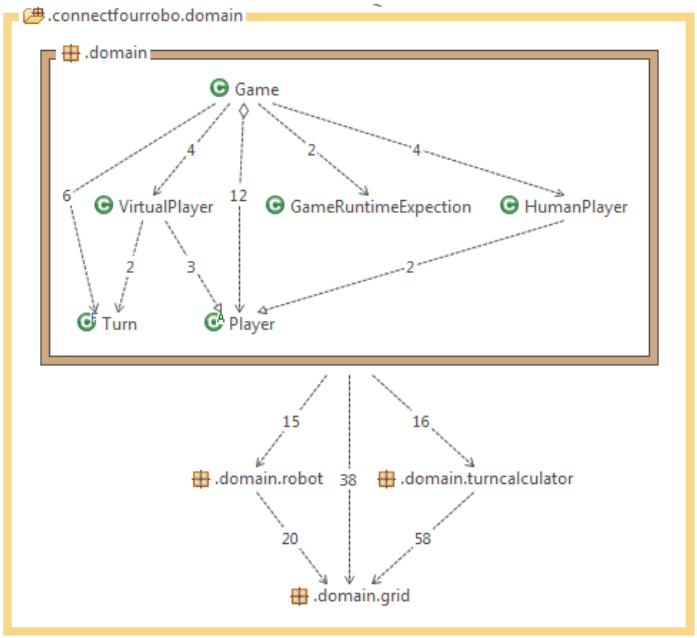


Abb. 14 Domain Package mit Verbindungen zu Subpackages

3.2.4 Klassen Übersicht und Implementationsentscheide

In diesem Kapitel wird auf spezifische Designelemente eingegangen. Die dargestellten UML-Diagramme sind dabei nicht vollständig, sondern stellen lediglich die relevanten Komponenten dar.

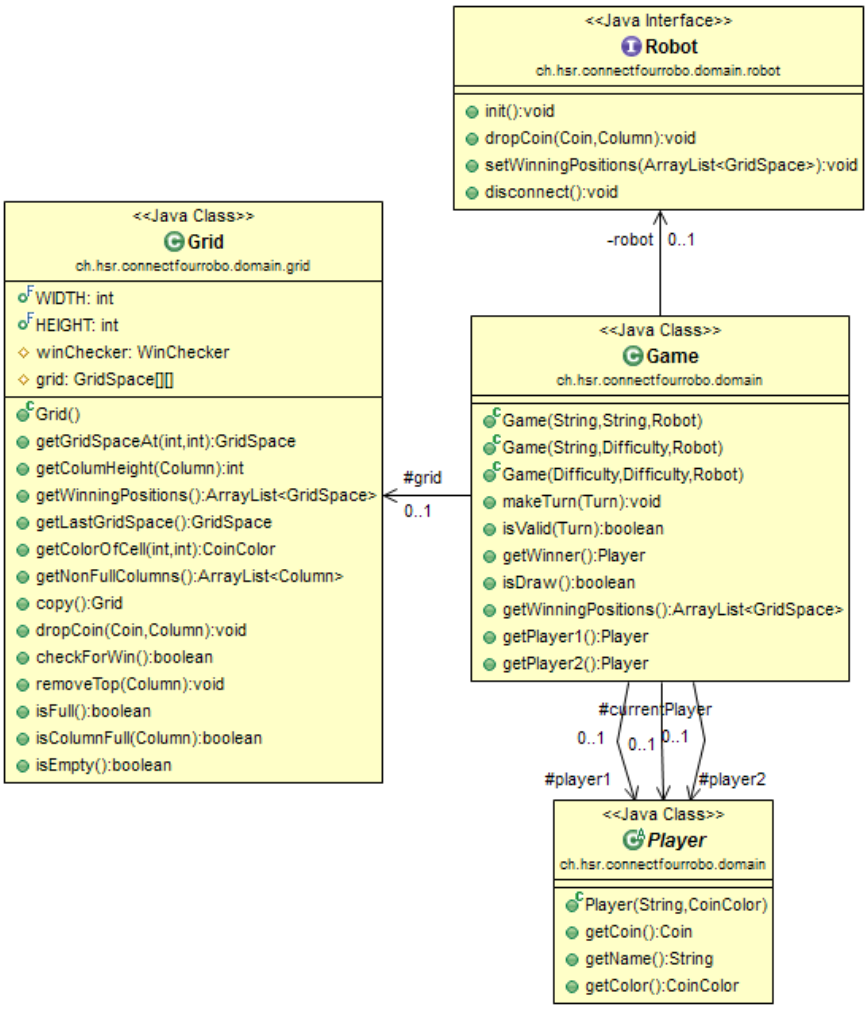


Abb. 15 Zentrale Klasse Games und deren Abhängigkeiten

3.2.4.1 *Klasse Game*

Die zentralste Klasse der Software stellt die Game-Klasse dar (siehe Abb. 15). Sie besitzt drei Konstruktoren um verschiedene Arten eines Spiels zu erstellen. Die möglichen Spiel-Modi sind: Spieler gegen Spieler, Spieler gegen KI, KI gegen KI (für Demo-Modus verwendet). Dazu nimmt der erste Konstruktor die zwei Spielernamen als Parameter entgegen. Ein Spiel mit einem KI kann demnach mit dem zweiten Konstruktor instanziiert werden, indem dem Konstruktor der Schwierigkeitsgrad der KI übergeben wird. Aus der Aufgabestellung ging hervor, dass es bei dieser Softwarelösung pro Spiel nur einen Roboter geben wird. Damit der Roboter trotzdem austauschbar bleibt, wurde entschieden den zu verwendenden Roboter als Parameter zu übergeben. Dadurch konnte eine einfachere Nutzung des Spiels mit einer Null-Implementation des Robot Interfaces erreicht werden. Damit musste die Implementation der Game-Klasse nicht angepasst werden. Auf die weiteren Methoden dieser Klasse wird nicht weiter eingegangen, da sie selbsterklärend sind.

3.2.4.2 *Klasse Grid*

Jedes Game erstellt ein eigenes Grid, auf welchem die Spielzüge (Turns) ausgeführt werden können. Ein Grid stellt eine Wrapper-Klasse um eine ArrayList von GridSpaces dar, um diese um zusätzliche Funktionen zu erweitern. Desweitern ist im Grid eine Instanz des WinCheckers hinterlegt, der beim Aufruf der checkForWin() – Methode anspringt, um einen Gewinner auf dem Spielfeld ausfindig zu machen. Weitere Informationen zum Thema WinChecker sind dem Kapitel 4.3.1 zu entnehmen.

3.2.4.3 *Abstrakte Klasse Player*

In jedem Game treten immer zwei Spieler gegeneinander an. Dabei sollte es keine Rolle spielen, ob es sich dabei um einen menschlichen Spieler oder um eine Künstliche Intelligenz handelt. Aus diesem Grund wurde eine abstrakte Klasse Player eingeführt.

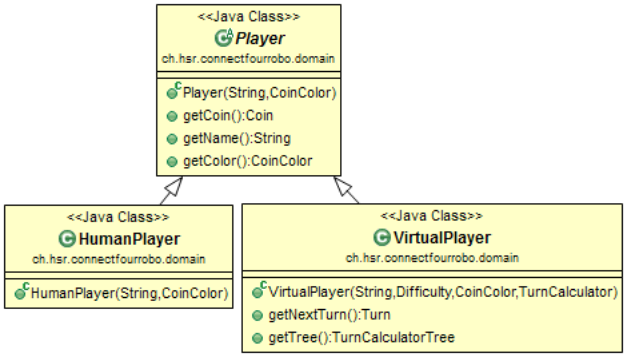


Abb. 16 Verschiedene Player Implementationen

Dabei stellt die Generalisierung „Player“ alle Methoden zur Verfügung, welche von den beiden abgeleiteten Klassen HumanPlayer und VirtualPlayer genutzt werden (siehe Abb. 16). Bei einem menschlichen Spieler handelt es sich um eine leere Klasse, die nur dazu verwendet wird um den menschlichen von einem Computergesteuerten Spieler zu unterscheiden.

Die Klasse VirtualPlayer stellt im Gegensatz zum HumanPlayer noch weitere Methoden bereit. Diese werden dazu verwendet um die Berechnung eines Zuges anzuwerfen, oder um der Visualisierung den Such-Baum des MiniMax-Algorithmus bereitzustellen.

3.2.4.4 *Klasse TurnCalculator*

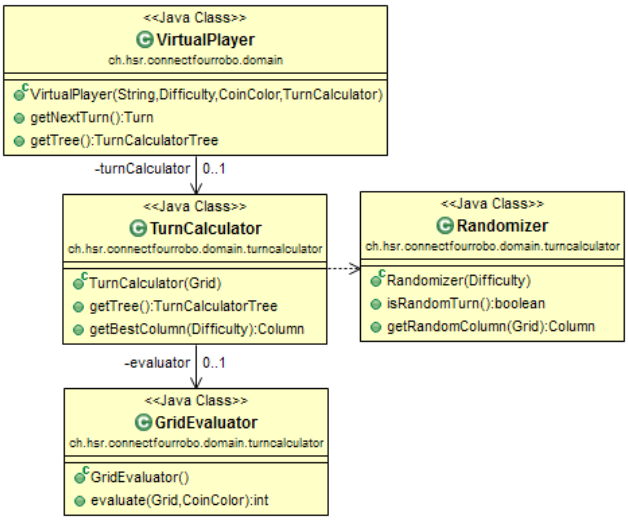


Abb. 17 Klassen zur Zugberechnung

Wird auf dem VirtualPlayer die Methode getNextTurn() aufgerufen, so holt sich der virtuelle Spieler beim TurnCalculator die erfolgversprechendste Spalte ab. Die Klasse wurde bewusst so designed, dass die Schwierigkeitsstufe, anstatt im Konstruktor, bei jedem Methodenaufruf mitgegeben werden muss. Dies hat den Vorteil, dass sich mit einem TurnCalculator mehrere Züge mit verschiedener „Güte“ berechnen lassen würden. Um die Ermittlung einer Spalte zu vereinfachen, stehen dem TurnCalculator Helferklassen zur Verfügung.

Bevor mit der Berechnung mittels MiniMax-Algorithmus begonnen wird, wird der Randomizer angeworfen, um sich mit einer vorkonfigurierten Wahrscheinlichkeit einen zufälligen Zug zu holen. Die jeweiligen Wahrscheinlichkeiten können im Config-File angepasst werden. (siehe Kapitel 7.3.5).

Wird kein zufälliger Zug gemacht, so muss ein bestmöglicher Zug mittels MiniMax Algorithmus gefunden werden. Auch an dieser Stelle wird der TurnCalculator von der GridEvaluator-Klasse, welche im Stande ist eine spezifische Spielsituation numerisch zu bewerten, unterstützt.

Weitere Infos zum Thema Zugberechnung sind dem Kapitel 4.3.3 zu entnehmen.

3.2.4.5 *Interface Robot*

Jedes Spiel wird immer auf einem Roboter gespielt. Da sich die Software aber auch ohne physikalischen Roboter testen lassen sollte, wurde ein Interface spezifiziert, welches die notwendigen Methoden definiert (siehe Abb. 18).

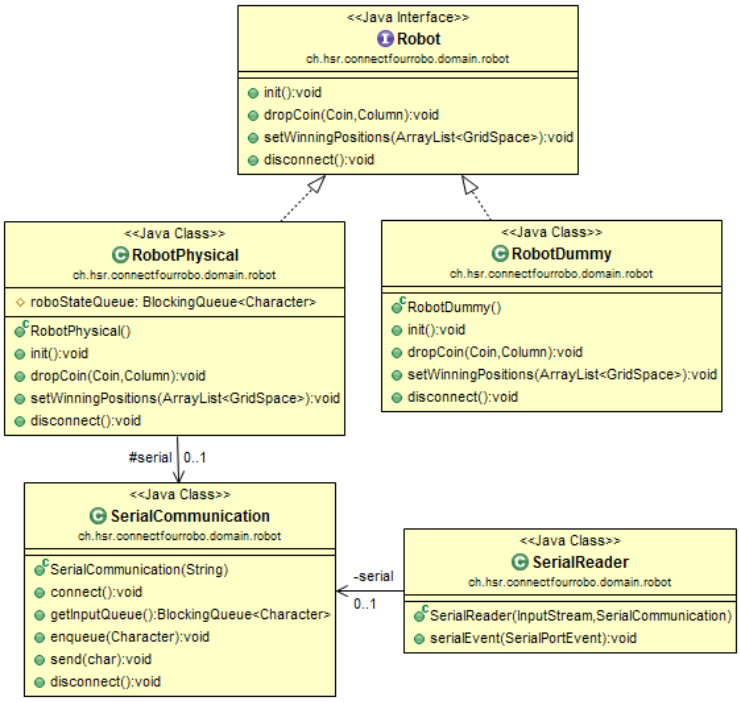


Abb. 18 Roboter Klassen mit Physical und Dummy Robot

Die Klasse RobotDummy dient nur zu Testzwecken und enthält dabei keinerlei Logik in den implementierten Methoden. Es wird lediglich ein Logeintrag erstellt, welcher festhält, dass die entsprechende Methode aufgerufen wurde. Diese Klasse kann verwendet werden, wenn z.B. keine serielle Schnittstelle vorhanden ist, oder wenn der Roboter-Teil für einen Test irrelevant ist. Des Weiteren kann per Konfigurationsfile sehr schnell auf den RobotDummy umgestellt werden, was bei einem Ausfall des Roboters an einem Infotag möglicherweise sehr nützlich sein könnte um immerhin das Vier Gewinnt ohne Roboter spielen zu können.

Weitaus wichtiger ist die Klasse RobotPhysical. Diese Klasse stellt die Repräsentation des realen Roboters dar. Mit einer Instanz dieser Klasse können Spielsteine eingeworfen und LEDs ein- und ausgeschaltet werden. Die Befehle werden dann an die SerialCommunication weitergeleitet. Ein wichtiges Element stellt die BlockingQueue innerhalb des RobotPhysicals dar. Von der Seriellen Schnittstelle empfangene Zustände werden in die BlockinQueue eingereiht. Mit der privaten Methode getRoboState() wird dann versucht einen empfangenen Zustand aus der Queue zu entfernen. Dies passiert durch die BlockingQueue blockierend, wobei der Vorteil ist, dass man nicht manuell pollen muss, um herauszufinden ob der Roboter bereits in einem neuen Zustand ist. Ausserdem kann somit gleich ein Timeout angegeben werden, nach dem der Roboter sicher geantwortet haben muss und sofern nötig kann dort auch gleich eine Exception geworfen werden. Somit konnte ein relativ komplizierter Ablauf in sehr wenigen Zeilen Code elegant gelöst werden.

3.2.4.6 *Robotersimulator*

Das Package Robotsimulation simuliert den physikalischen Roboter und stellt ein eigenständiges Programm dar. Wird seitens des Servers eine Instanz der Klasse PhysicalRobot verwendet, muss entweder mit dem richtigen Roboter oder mit dem Robotersimulator kommuniziert werden. Möchte auf die Verwendung des Robotersimulators verzichtet werden, muss seitens des Servers das Objekt DummyRobot, welches ein Null-Implemented-Object darstellt, verwenden werden.

Über das Main im RobotSim-Starter wird ein Thread von RobotSimRespondern angeworfen. Dieser reagiert auf seriellen Input, welcher vom Objekt RobotPhysical seitens des Servers kommt. Die Klasse RobotSimState repräsentiert die vier Zustände in welchem sich der physikalische Roboter befinden kann (siehe Kapitel 4.2.6 und Kapitel 4.4.1). Die Klasse RobotSimWorkPeriod ist ein Thread welcher von RobotSimResponder angeworfen wird sobald mechanische Arbeit simuliert werden soll. Während dieser Zeit wird der Status des Robotersimulator auf Busy gestellt. Dies wurde so umgesetzt, da der Roboter zu jedem Zeitpunkt antworten muss.

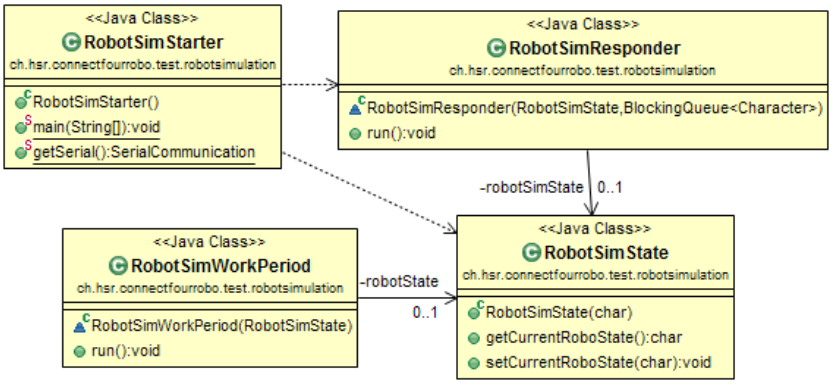


Abb. 19 Unabhängige Klassen zur Simulation des Roboters

3.2.4.7 *Klasse Config*

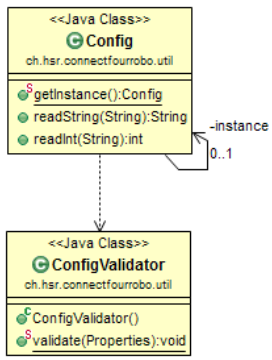


Abb. 20 Config Klassen

Um das Lesen der Konfigurationsdatei zu vereinfachen, wurde die Klasse Config als Singleton implementiert. (siehe Abb. 20) Diese Klasse stellt zwei identische Methoden mit unterschiedlichem Rückgabetyp zur Verfügung. Dies hat den Vorteil, dass die Typprüfung zentral in einer Klasse gehandhabt werden kann.

Des Weiteren wird beim ersten Aufruf der Config Klasse die Konfiguration mittels ConfigValidator validiert. Dieser überprüft, ob alle Properties in der Datei vorhanden sind und ob sich deren Werte im erlaubten Bereich befinden.

3.2.5 Ablauf Spielerstellung und Spielzüge

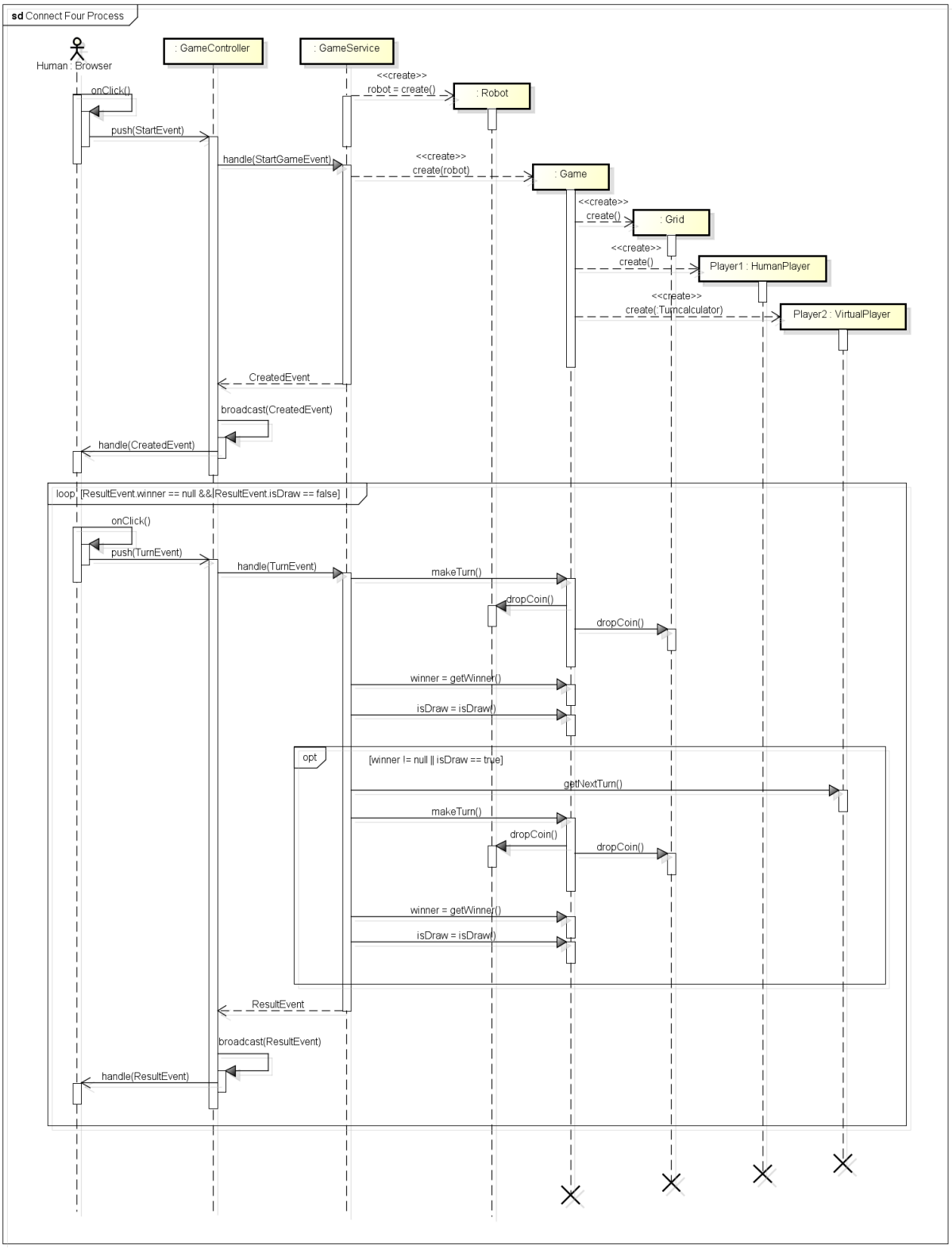


Abb. 21 Sequenzdiagramm eines Spielablaufs

Dieses Sequenz Diagramm (siehe Abb. 21) zeigt den Ablauf eines Spiels mit einem virtuellen Gegner auf. Wird der Ablauf gegen einen virtuellen Spieler verstanden, ist dieser gegen einen Human Player schnell zu verstehen.

Zu Beginn erzeugt das Game Objekt die Spieler Objekte welche im weiteren Verlauf in einem CreatedEvent zurückgegeben werden. Weiterführend wird der CreatedEvent im JSON Format an den Browser übermittelt.

Aus Komplexitätsgründen bei der Darstellung wird das Sequenz Diagramm nach der Schleife nicht detailliert dargestellt. Nach der Schleife würde der Gewinner noch gesetzt werden, sowie die LEDs des Roboters angeschaltet an denen sich die vier Gewinnsteine befinden.

Zu beachten ist, dass die Instanzen der Klassen GameService, GameController und Roboter einmal zu Beginn erstellt werden. Das Game sowie deren Objekte werden jedes Mal neu instanziiert. Methoden auf dem Controller können durch HTTP Requests asynchron ständig aufgerufen werden. Methoden des GameService welche vom Controller benutzt werden sind mit Java synchronized synchronisiert. Somit wird sichergestellt, dass auf dem GameService keine Nebenläufigkeiten auftreten. Auf dem GameService gibt es weiter eine Semaphore welche relevant für zwei Methoden ist. Die eine Methode behandelt den Ablauf des Virtuellen Spiels sofern niemand am Roboter spielt. Dabei wird bei jedem nächsten Zug die Semaphore abgeholt. Die zweite Methode die Gebrauch der Semaphore macht, wird aufgerufen falls sich ein neuer Spieler verbunden hat und ein Spiel starten möchte. Somit ist sichergestellt, dass ein Virtuelles Spiel wirklich abgebrochen werden kann und das Programm danach nicht in einen ungewollten Zustand ist.

3.2.6 Roboter Kommunikation

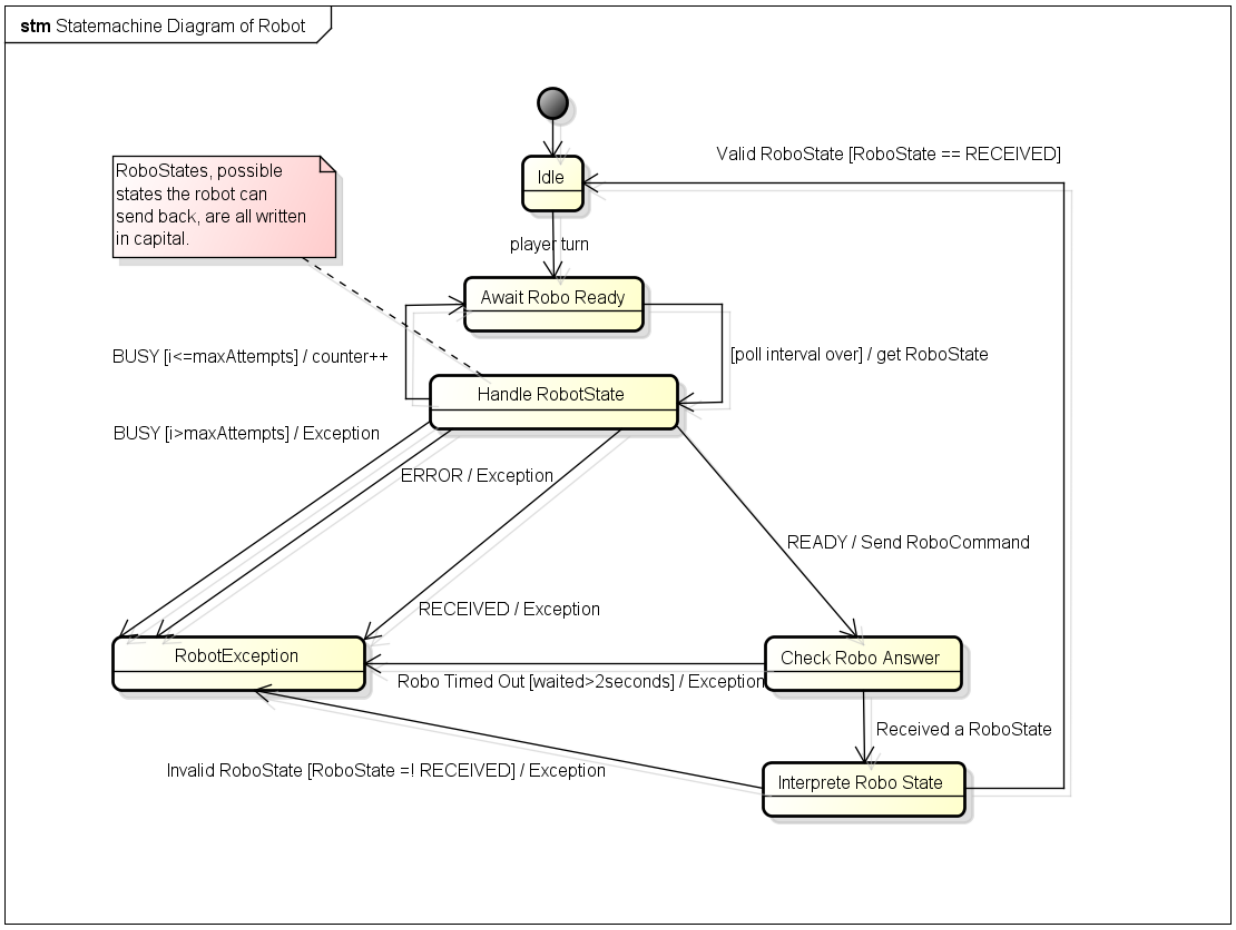


Abb. 22 Kommunikationszustände Roboter-Interface zu Physischen-Roboter

Der Roboter und somit das Roboterinterface war bereits aus einer vorgängigen Arbeit gegeben. An dieser Stelle ist es wichtig zu erwähnen, dass wir keinen Einfluss auf das Design der Roboterschnittstelle hatten. Aus diesen Gründen war die Kommunikation zum Roboter bereits sehr eng vorgegeben.

Leider war die Schnittstelle, und daher die Kommunikation zum Roboter spärlich und vereinzelt inexistent dokumentiert. Deshalb mussten am Anfang, nach Absprache mit dem Roboterbauer und der IFS Assistenten, sowie nach Prüfung der vorhandenen Dokumentation einige Tests durchgeführt werden (siehe Kapitel 4.4.1), um das Verhalten des Roboters zu verstehen. In einem zweiten Schritt konnte dann in Eigenarbeit mit einigem Aufwand ein Zustandsdiagramm zur Kommunikation mit dem Roboter entworfen werden (siehe Abb. 22).

Zum Verständnis des Zustandsdiagramms ist es wichtig zu wissen, dass der Roboter generell über vier verschiede Zustände verfügt. Die Zustände sind: READY, BUSY, ERROR und RECEIVED. Das Protokoll also die somit möglichen Befehle sowie die Regeln zur Schnittstelle des Roboters können aus unserem Kapitel Testing (siehe Kapitel 4.4.1) abgeleitet werden.

3.2.7 Serverseitige Implementation

Serverseitig wurde das SpringSource Framework, ein Open Source Java Framework verwendet, welches die Entwicklung von Java EE-Applikationen vereinfacht. Besonders die Erweiterung Spring MVC erlaubt eine saubere Trennung von Darstellung und Logik. Durch die zusätzlich von Spring bereitgestellten Annotationen ist es möglich den Code schlank und übersichtlich zu halten. Explizit wurden folgende Annotationen verwendet:

- @Controller wurde in der GameController Klasse verwendet um dem SpringSource Framework zu signalisieren, dass es sich bei dieser Klasse um eine URL Mapping Klasse handelt.
- Zusätzlich zur Controller Annotation benötigt es über den erforderlichen Methoden die @RequestMapping(value="url/path", method="GET oder POST") Bezeichnung um den Pfad und die Art des HTTP-Aufrufs zu bestimmen.
- Mit dem Begriff @Autowired konnte die Service Klasse übergreifend und automatisch instanziiert dem Controller bekannt gemacht werden.
- Die GameService Klasse wurde mit der Annotation @Service gekennzeichnet womit sie von Spring als Singleton instanziiert wird und wie zuvor genannt mit @Autowired in andere Klassen bzw. Objekte eingebunden werden kann.
- Des Weitern enthält die GameService Klasse die Annotation @Scheduled(fixedDelay=5000). Damit lässt sich eine Methode alle 5 Sekunden ausführen. Dies wurde verwendet um zu prüfen ob die Spieler längere Zeit (es wurden 40 Sekunden definiert) inaktiv waren und somit vom Spiel ausgeschlossen werden können. Ausserdem wurde dieses Hilfsmittel verwendet um den Demo-Modus zu starten.

Im Gegensatz dazu wurde die alte Studienarbeit ohne den Einsatz eines unterstützenden und stabilen Frameworks entwickelt. Es wurde die Basisversion von Java EE verwendet. Im Vergleich zur heutigen Software Lösung, war die Aufteilung in Model-View-Control praktisch nicht vorhanden.

3.2.8 Server – Client Kommunikation

Das folgende Diagramm soll die Kommunikation zwischen Server und Client in einer vereinfachten Darstellung erläutern. Um den Event basierten Ablauf des Programms möglichst übersichtlich zu halten wurden auf die herkömmlichen Sequenzdiagramm Vorgaben verzichtet. Zur Kommunikation wird auf das Atmosphere Framework zurückgegriffen welches die Kommunikation über einen Websocket Channel bzw. eine Fallback Variante, in unserem Fall Streaming, zur Verfügung stellt. Um die Kommunikation zwischen Browser und Server korrekt zu gewährleisten, muss der Kommunikationskanal per JavaScript mit Hilfe von jQuery und Atmosphere Plugin aufgebaut werden.

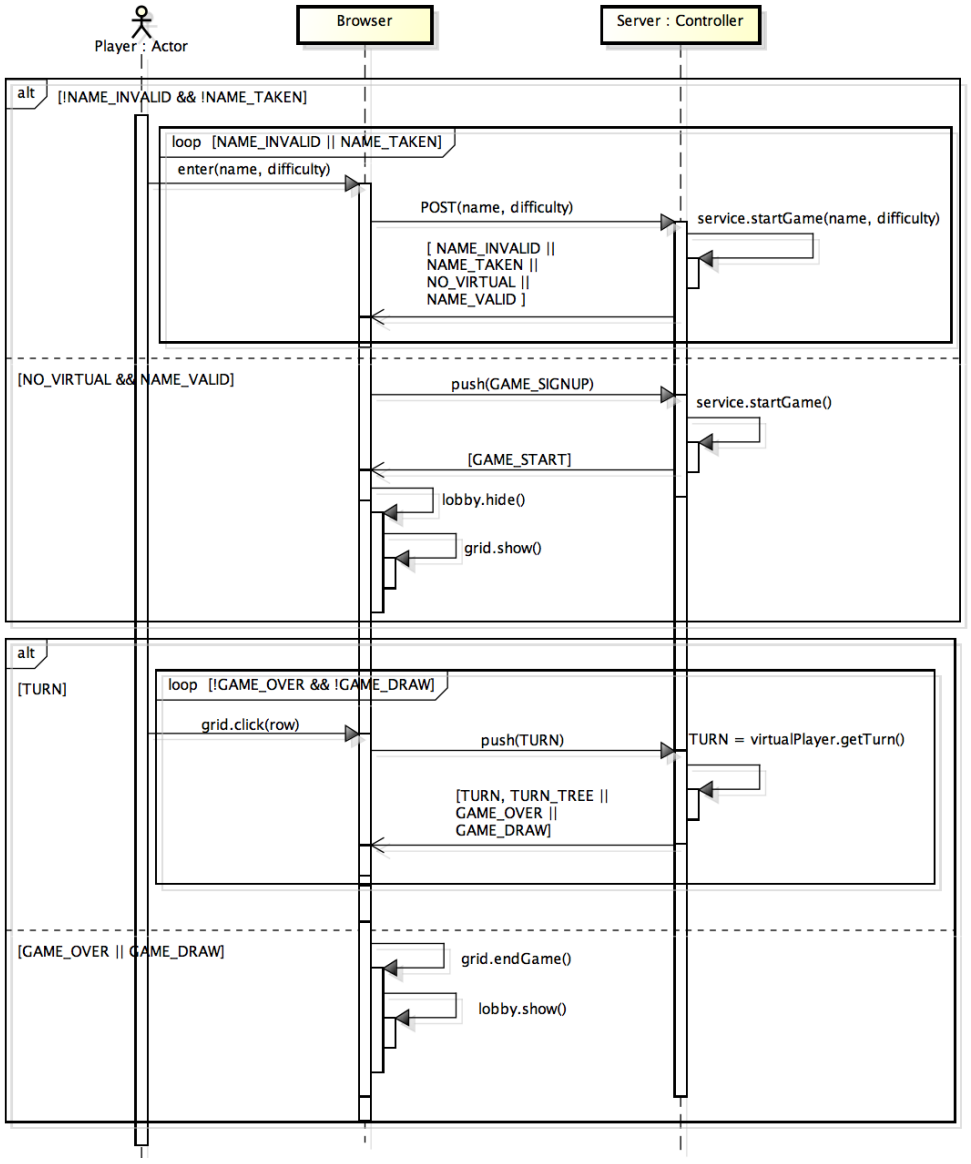


Abb. 23 Sequenzdiagramm der Kommunikation zwischen Browser und Server

Aus dem Diagramm ist ersichtlich, dass sich der Ablauf in vier Hauptbereiche unterteilt.

1. Der Benutzer gibt seinen Namen ein (erlaubt sind 3 - 15 Zeichen, a-z, A-Z, 0-9, _ und -) und wählt nach Belieben einen Schwierigkeitsgrad. Die Daten werden als HTTP POST Request wie folgt verpackt und an den Server übermittelt.

   HTTP Type: POST

   URL: .../register?difficulty= [(kein Argument) || EASY || MEDIUM || HARD]

   HTTP Request Body: { ‚name': ‚Spielername', ‚coinColor': ‚' }

Der Server validiert den angegebenen Namen und den Schwierigkeitsgrad und gibt je nach Auslastung oder Unstimmigkeiten folgende GameEventType zurück:

- NAME_VALID: Der angegebene Name entspricht den Vorgaben und ein PlayerSlot ist frei
- NAME_INVALID: Name entspricht nicht den Konformitäten
- NAME_TAKEN: Ein zweiter Spieler hat bereits den gleichen Namen
- NO_VIRTUAL: Es wurde eine Schwierigkeitsstufe gewählt, jedoch wartet schon ein menschlicher Gegenspieler auf einen Mitspieler. (Der Spieler hat nun die Möglichkeit die Herausforderung anzunehmen oder zur Lobby zurückzukehren)

Der Spieler bleibt so lange im ersten Bereich bis entweder ein NAME_VALID Event vom Server zurückkommt oder der Besucher nach einem NO_VIRTUAL die Herausforderung annimmt und gegen den Mitspieler spielen möchte.

2. Nachdem der Spieler beim Server registriert wurde, sendet der Browser über den zu Beginn aufgebauten Channel einen GAME_SIGNUP Event an den Server.

   HTTP Type: POST
   URL: .../channel
   HTTP Request Body: {‚type': ‚GAME_SIGNUP', ‚data': „{ ‚name': ‚Spielername', ‚coinColor': ‚' }"}

   Nach Erhalt eröffnet der Server ein Spiel und informiert alle beteiligten Parteien über den Channel mit einem GAME_START Event, dass das Spiel gestartet wurde. Im Browser der Spieler wird die Lobby ausgeblendet und das Spielfeld (Grid) eingeblendet.

3. Der Spieler der am Zug ist, hat nun die Möglichkeit auf die Spalte zu klicken in welcher er einen Stein versenken möchte. Der Klick Event schickt einen TURN Event an den Server, welcher den Spielzug weiterverarbeitet.

   HTTP Type: POST
   URL: .../channel
   HTTP Request Body:
   turn = { 'coin': { 'coinColor': coinColor }, 'column': column };
   { 'type': 'TURN', 'data': JSON.stringify(turn) };

   Falls kein Computergegner vorhanden ist, wird der Spielzug über den Channel an alle Parteien propagiert. Ist ein Computergegner registriert, wird dessen Zug berechnet und in ein JSON-String umgewandelt und an den Spieler gesendet.
   Diese Schritte werden solange wiederholt bis vom Server ein GAME_OVER oder GAME_DRAW gesendet wird.

4. Falls zuvor ein GAME_OVER oder GAME_DRAW empfangen wurde, wird dem Spieler eine entsprechende Nachricht präsentiert und auf die Lobby zurück geleitet. Bei einem GAME_OVER Event wird zusätzlich ein GAME_WINNER Event angehängt um dem Spieler mitzuteilen ob er gewonnen oder verloren hat. Des Weiteren sind in einem GAME_OVER Event die Gewinn-Positionen verpackt um diese auf dem Spielfeld zu animieren.

3.2.8.1 *Fazit*

Der Vorteil dieser Lösung ist, dass die Kommunikation ausschliesslich auf Events basiert. In der alten Lösung wurden das Spielverhalten und das Ändern der Oberfläche komplett mit der eigentlichen Server Applikation verschweisst, wodurch die Wartung erheblich erschwert wurde. Des Weiteren ist es mit der neuen Kommunikationslösung möglich das GUI komplett zu ersetzen ohne das Änderungen am Java Code nötig sind. Dies ist natürlich auch umgekehrt der Fall. Denn solange die Response Events gleich bleiben, kann die Server Applikation komplett ersetzt werden, ohne die Funktionalität des Frontend einzuschränken.

3.2.9 Benutzeroberfläche und clientseitige Implementationen

3.2.9.1 *Kommunikation mit Server*

Zur Kommunikation zwischen Client und Server wird auf die neue Technologie Websocket gesetzt. Da das in dieser Arbeit verwendete Spring Framework in der aktuellen Version keine WebSockets unterstützt, wurde zur Unterstützung das schlanke Atmosphere Framework beigezogen. Dabei handelt es sich um ein portables Websocket Framework mit Java Support und bereits implementierten Fallback Varianten (wie Streaming oder Long-Polling). Durch die einfachere Nutzung von WebSockets konnte die Kommunikation erheblich vereinfacht und verbessert werden. Als Vergleich zur alten Software Lösung: In der alten SA wurden keine WebSockets verwendet, sondern ein selbstgebasteltes „Long-Polling“, welches sehr instabil lief und zu ungewollten Nebenläufigkeiten führte.

3.2.9.2 *Implementation der Spieloberfläche*

Zur vereinfachten JavaScript Programmierung wurde die beliebte Library jQuery verwendet. Damit lassen sich Mutationen des DOM-Baumes, Event Handling, Animationen des Spielfeldes etc. erheblich vereinfachen. Das in dieser Arbeit entwickelte jQuery Plugin wurde in drei einzelne JavaScript Dateien unterteilt:

1. *jquery.connectfour.application*: Dieser Teil des Plugins ist für die Kommunikation mit dem Server und der Abhandlung der Events vom Server zuständig. Des Weiteren fungiert es als Controller des jQuery View Layers.
2. *jquery.connectfour.view*: Mit Hilfe dieses Bereichs kann ein Spielfeld in ein HTML-Element gezeichnet werden. Ausserdem ist es für die Animation der fallenden Steine zuständig. Weiter zeichnet es die gespielten Steine in das zuvor gezeichnete Feld.
3. *jquery.connectfour.util*: Hier werden Hilfsmethoden definiert die auch unabhängig von diesem Projekt verwendet werden können.

Auf der Client Seite wurde das Design mit Hilfe des CSS-Framework YAML als Responsive Design umgesetzt. Die Umsetzung des Spielfelds mittels des Responsive Ansatz, war zu Beginn nicht ganz einfach, konnte aber nach einiger Recherche mit dem Ansatz von „Fluid Squares v2“ [1] gelöst werden. Die entscheidenden Vorteile und Verbesserungen bzw. Änderungen gegenüber der alten Lösung werden in den folgenden Abschnitten näher erläutert.

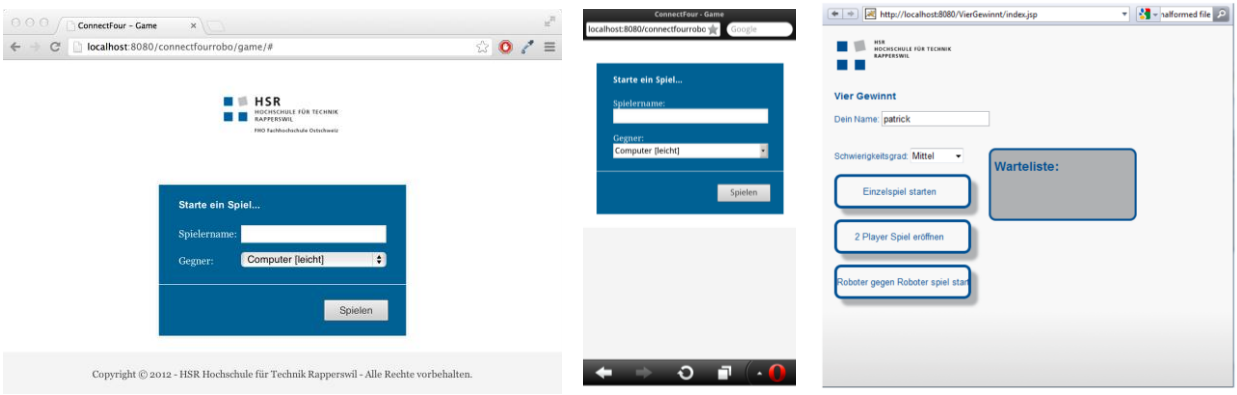


Abb. 24 Neues GUI im Browser auf einem Mobile Device, sowie das alte GUI

[1] (Macdonald 2011)

In der Spiel Lobby wird der Benutzer aufgefordert einen Spielernamen anzugeben und falls ein Computergegner gewünscht ist, dessen Schwierigkeitsgrad zu wählen. Dabei ist die Eingabe wie zuvor im Kapitel 4.2.8 bereits erwähnt beschränkt. Das beschränken des Namens bewirkt unter anderem ein verhindern von CrossSiteScripting und vereinfacht die Handhabung des Spielernamens im Code. Die zwei grössten Änderungen bzw. Verbesserungen gegenüber der alten Lösung sind die zwingende Angabe des Namens und der Einführung eines Auswahlmenüs der Schwierigkeitsstufen bzw. Mitspieler mit einem expliziten Start-Button. Zum Vergleich: In der alten Lösung (siehe Abb. 24 rechts) war das Namensfeld kaum sichtbar , die Wahl der Schwierigkeitsstufe wurde unklar dargestellt und ein Spiel konnte je nach Bedürfnis über unterschiedlichen Buttons gestartet werden.

Abb. 25 Warte Dialoge 1 und 2

Im Projekt wurde entschieden, keine Warteliste einzuführen. Wie in Abbildung 25 links ersichtlich, wird beim Start eines Spiels gegen einen menschlichen Spieler eine Wartemeldung angezeigt. Falls bereits ein Spiel gestartet wurde, wird wie Abbildung 25 rechts zeigt eine Meldung angezeigt. Ausschlaggebender Aspekt zur Entscheidung für diese Lösung war, dass sich die Personen die nicht spielen das Spielgeschehen auf dem Roboter verfolgen.

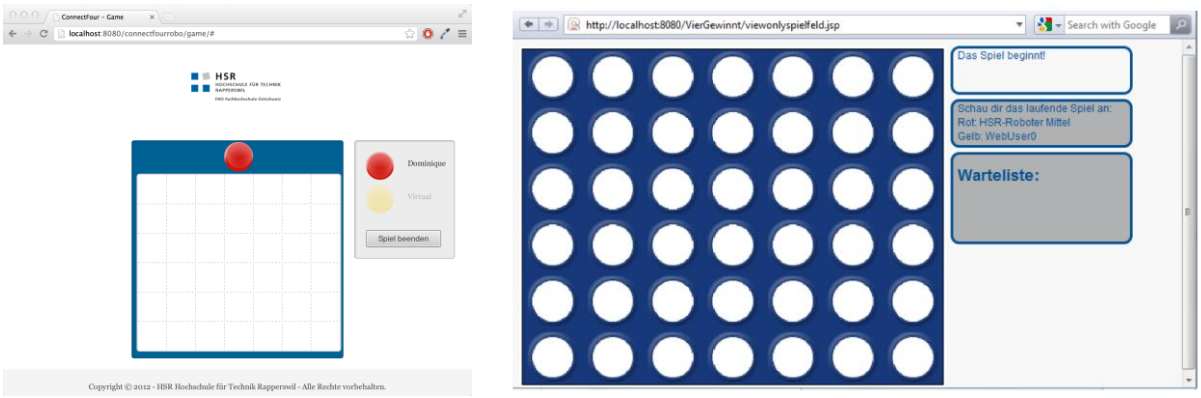


Abb. 26 Darstellung des neuen (links) und alten (rechts) Spielfelds

Die Darstellung des Spielfelds und die Steuerung des Spiels (siehe Abb. 26) wurde insofern intuitver gestaltet, dass der Spielstein nur mit einer Mausbewegung bewegt und mit einem Klick auf eine beliebige Spalte „fallen gelassen“ werden kann. In der Vorgänger Version musste erst die Spalte mit einem Klick markiert und danach mit einem zweiten Klick bestätigt werden.

Abb. 27 Darstellung auf Mobile Devices. neu (links) und alt (rechts)

Auf den Mobilen Geräten wird das Spielmenü wie in der alten Version unterhalb des Spielfelds dargestellt. Auch hier wurde die User Experience verbessert. Es muss nur noch eine Berührung der gewünschten Spalte stattfinden um den Spielstein in die gewünschte Spalte fallen zu lassen. In der alten Version wurde zu erst mit einer Berührung die Spalte angewählt und mit einem kleinen Pfeil gekennzeichnet, dann musste mit einer zweiten Berührung die gewählte Spalte bestätigt werden. Die zweite Berührung, so der Roboterbauer, war vielen Spieler nicht von Beginn weg klar, wodurch der Spielfluss beeinträchtigt wurde.

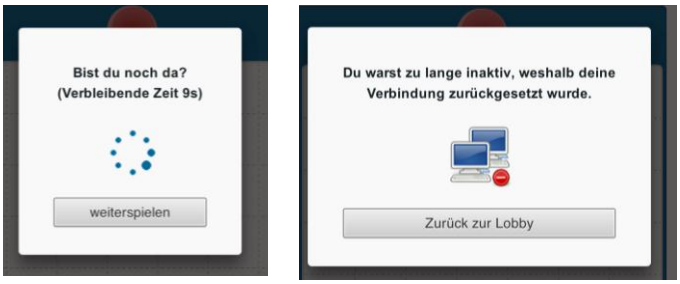


Abb. 28 Dialoge bei Spieler Timeout

Bei Inaktivität wird dem Spieler nach Ablauf des Timeouts von 30 Sekunden eine Aufschiebungsanfrage (siehe Abb. 28 links) angezeigt, falls der Spieler nicht innerhalb der gegebenen 10 Sekunden antwortet wird die Verbindung nach insgesamt 40 Sekunden zurückgesetzt (siehe Abb. 28 rechts).

Nach einem Sieg oder einer Niederlage wird dem Benutzer die entsprechende, in Abbildung 29 aufgezeigte Meldung zurückgegeben.

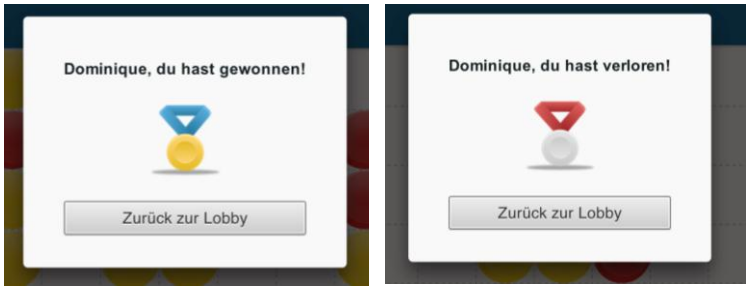


Abb. 29 Dialoge bei Sieg bzw. Niederlage

Folgende Fehlerbehandlungsmeldungen gibt es:

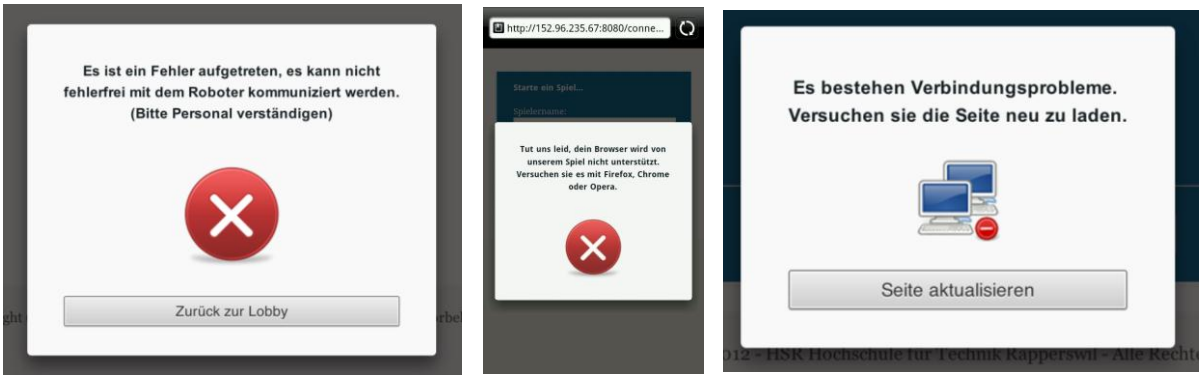


Abb. 30 Mögliche Fehlermeldungen (von links): Roboterfehler, nicht unterstützter Browser und Verbindungsprobleme

3.2.9.3 *Implementation der Algorithmus Visualisierung*

Um den Spielalgorithmus einfach zu visualisieren wurde das JavaScript InfoVis Toolkit (JIT) verwendet. Mithilfe von JIT lassen sich bequem Suchbäume beliebiger Grösse aufbauen. Die Information zur Visualisierung werden wie zuvor mit Hilfe von WebSockets vom Server als TURN_TREE Event verpackt und übermittelt.

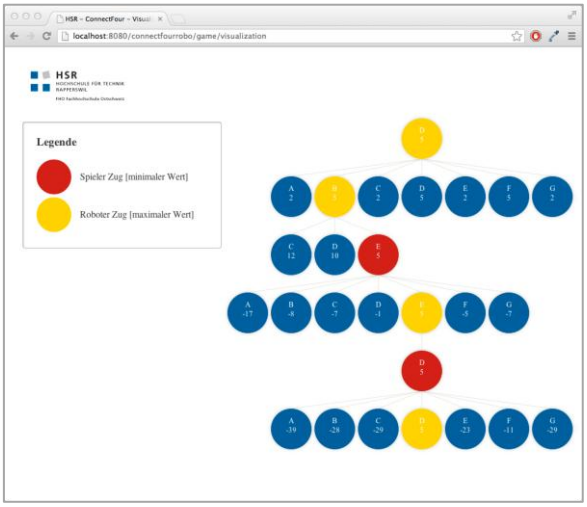


Abb. 31 Visualisierung des Spielalgorithmus

Die Visualisierung wurde wie bereits erwähnt auf Wunsch des Auftraggebers in Form eines Baums gestaltet, wobei jeweils die Äste geöffnet werden, welche zur Spielzugentscheidung benutzt wurden. Des Weiteren wurden wie in Abbildung 31 ersichtlich der Übersichtlichkeit halber die gewählten Züge mit den Spielerfarben markiert. In den einzelnen Knoten wird der berechnete Wert für den Zug dargestellt.

3.2.10 Fehlerbehandlung

3.2.10.1 *Verbindungsfehler zwischen Client und Server*

Bei einem Netzwerk-Verbindungsproblem wird standardmässig (Atmosphere jQuery-Plugin) versucht die Verbindung automatisch wiederherzustellen (dabei wird der Standardwiederholungswert verwendet siehe Website des Herstellers https://github.com/Atmosphere/atmosphere). Falls die Verbindung nicht mehr hergestellt werden kann, wird dem Benutzer eine entsprechende Meldung präsentiert. Die Meldung empfiehlt dem Benutzer die Website erneut zu Laden.

3.2.10.2 *Roboter Kommunikationsfehler*

Für alle Fehler die bei der Kommunikation mit dem Robot auftreten können, werden entsprechende Exceptions geworfen. Diese werden bis in die Serviceklasse hochgeworfen, dort wird versucht die Kommunikation mit dem Roboter wieder aufzunehmen. Falls dies nicht klappt wird ein Eintrag im Logfile erfasst und dem Benutzer auf dem Webinterface eine passende Nachricht über die Verbindungsprobleme angezeigt. Danach kann der Benutzer über die Spiel Lobby versuchen ein neues Spiel zu starten. Bei jedem Neustart wird versucht den Roboter wieder anzusprechen, falls dies nicht funktioniert wird dem Benutzer wieder die Fehlermeldung angezeigt und er kann es nochmals probieren. Die Software kann somit nicht aufgrund eines Kommunikationsproblems abstürzen, wie dies in der alten Lösung teilweise der Fall war.

3.2.10.3 *Fehler im Config File*

Sobald das .war File neu deployed wird, wird überprüft, ob das Config File Fehlerhaft ist oder nicht. Tritt dabei ein Fehler auf, so lässt sich das .war File nicht platzieren und es wird eine Exception angezeigt. Änderungen am Config File werden immer nur wirksam, wenn das .war File neu in den Tomcat Server geladen wird. Wie sich das Config File ändern lässt und in welchem Ordner die Datei liegt, ist dem Kapitel 7.3.5 zu entnehmen.

3.3 Algorithmen

3.3.1 WinChecker Algorithmus

Ziel eines WinCheckers ist es das Spielfeld zu analysieren und zu melden, ob eine Gewinnsituation vorliegt. Diese Aufgabe hat vor unserer Arbeit das Grid selbst übernommen. Das Prinzip von „High Cohesion“ wurde dabei verletzt. Um die Kohäsion zu steigern und das „Coupling“ möglichst tief zu halten, wurde ein separater „Winchecker“ implementiert.

In einem zweiten Schritt wurde der Algorithmus an sich verbessert.

3.3.1.1 *Vorher:*

Von unseren Vorgängern wurde ein sehr simpler Algorithmus implementiert. Es wurden der Reihe nach alle Horizontalen, Vertikalen und Diagonalen durchlaufen um nach einem Gewinner zu suchen (siehe Abb. 32).

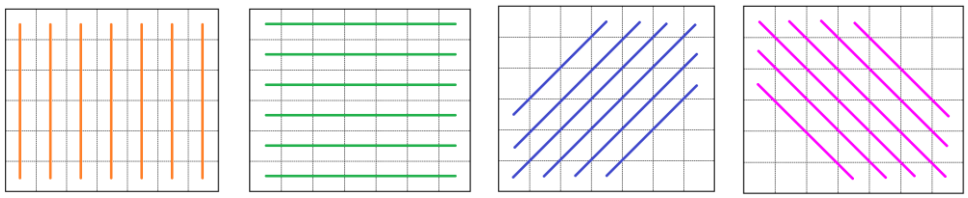


Abb. 32 Durchlauf aller Vertikalen, Horizontalen und Diagonalen

Dabei werden im schlechtesten Fall 49+49+37+37 = **172** Felder geprüft. Ausserdem würde bei einem Spielfeld von nicht konstanter Grösse die Berechnung mit einer Laufzeit von $O(4n^2)$ skalieren. Da aber die Spielfeldgrösse immer gleich ist (bedingt durch den Roboter), spielt diese Skalierung eine untergeordnete Rolle.

3.3.1.2 *Nachher:*

Um die Performance zu erhöhen, haben wir uns dafür entschieden, weitere Informationen in die Berechnung miteinzubeziehen: die Position und Farbe des zuletzt eingeworfenen Steines. Da die Gewinnprüfung sowieso nach jedem Zug durchlaufen werden muss, kann nur derjenige Spieler gewonnen haben, welcher den letzten Stein eingeworfen hat. Ausserdem kann der Gewinn auch nur rund um die letzte Position erfolgt sein. Somit lassen sich die Anzahl Vergleiche drastisch reduzieren.

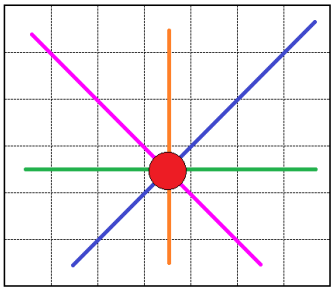


Abb. 33 Es werden nur noch die Relevanten Zellen analysiert

In Abbildung 33 ist zu sehen, welche Felder noch analysiert werden müssen. Auf allen anderen Feldern kann kein Gewinn vorliegen, da der letzte Stein (rot) nicht im Subset jener Gewinnkombination enthalten ist.

Mit dieser Methode müssen weitaus weniger Vergleiche gemacht werden. Im schlechtesten Fall werden 6+7+6+6 = **25** Felder geprüft. Das ist fast 1/6 der Anzahl Vergleiche des alten Algorithmus. Des Weiteren würde dieser Ansatz auch bei variablen Spielfeldgrössen besser skalieren.

3.3.1.3 *Fazit:*

Obwohl auf modernen Rechnern auch der alte Algorithmus in Sekundenbruchteilen durchläuft, stellt die neue Variante dennoch eine wesentliche Verbesserung dar. Der WinChecker wird nämlich auch von der KI verwendet um den bestmöglichen Zug zu finden. Je nach Schwierigkeitsgrad kann der WinChecker einige Hunderttausende Male aufgerufen werden. Somit wird auch der Geschwindigkeitsbonus der überarbeiteten Version um den entsprechenden Faktor erhöht.

Ein kluger WinChecker ist die Grundvoraussetzung eines performanten TurnCalculators (derjenige Teil der KI, welcher den nächstbesten Zug generiert).

3.3.2 GridEvaluator Algorithmus

Um die KI dazu zu bewegen einen möglichst guten Zug vorzuschlagen muss erst einmal definiert werden, was ein guter Zug ist. Es ist natürlich einleuchtend, dass jene Züge besser sind, welche zu einer Spielsituation führen die entweder einen Sieg zur Folge haben oder zumindest die Möglichkeit auf diesen erhöhen. Dabei sollen natürlich auch die Chancen des Gegners auf einen Sieg möglichst klein gehalten werden. Dies alles ist die Aufgabe des GridEvaluators.

Der Algorithmus wurde so entworfen, dass dieser eine Spielsituation aus der Sicht des sich am Zuge befindenden Spielers bewertet. Alles was er dazu benötigt, ist dessen Farbe und das Spielfeld (Grid).

3.3.2.1 *Funktionsweise*

Damit berechnet werden kann, für welchen Spieler eine gewisse Spielsituation vorteilhafter ist, wird als erstes das gesamte Spielfeld in alle möglichen Gewinnkombinationen zerteilt(siehe Abb. 34). Eine Gewinnposition wird danach durch einen Block von vier Zellen repräsentiert.

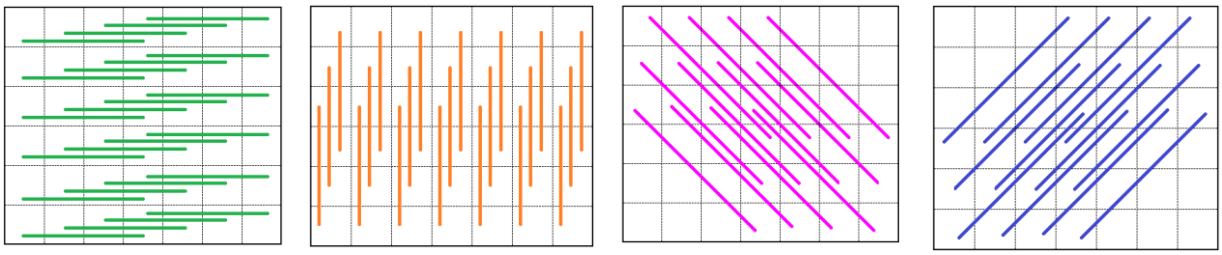


Abb. 34 Markierungen an welchen Positionen eine Viererreihe zustande kommen kann

Danach wird jeder 4er Block nach den folgenden Kriterien bewertet (aus der Sicht von Rot):

1 roter Stein und 3 leere Felder	2 rote Steine und 2 leere Felder	3 rote Steine und 1 leeres Feld	4 rote Steine
+1 Punkt	**+10 Punkte**	**+100 Punkte**	**+1000 Punkte**

1 gelber Stein und 3 leere Felder	2 gelbe Steine und 2 leere Felder	3 gelbe Steine und 1 leeres Feld	4 gelbe Steine
-1 Punkt	**-10 Punkte**	**-100 Punkte**	**-1000 Punkte**

Alle anderen Kombinationen ergeben 0 Punkte. Dies gilt auch für den Fall, wenn eine Kombination unterschiedlich farbige Steine enthält. In solch einem Fall kann an dieser Position keiner der

beiden Spieler eine Viererkette erreichen. Wird aus der Sicht von Gelb bewertet, wird das Ergebnis noch negiert (alle Gewinnkombinationen von Rot werden negativ, jene von Gelb positiv).

Summiert man alle Bewertungen der Viererblöcke auf, so ergibt sich daraus eine Gesamtbewertung für das Spielfeld. Mit dieser Methode kann eine Spielsituation objektiv bewertet werden.

Hier noch ein erweitertes Beispiel:

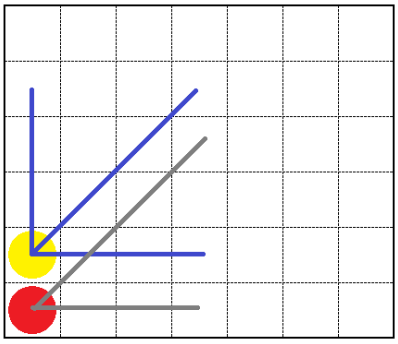


Abb. 35 Zwei Spielsteine und deren Potenzial noch zu einer Viererreihe zu werden

Dieses Beispiel würde sich den Wert aus der Sicht von Rot wie folgt berechnen:
Rot hat horizontal schon einen von vier Steinen -> **+1 Punkt**
Rot hat diagonal schon einen von vier Steinen -> **+1 Punkt**
Gelb hat horizontal schon einen von vier Steinen -> **-1 Punkt**
Gelb hat diagonal schon einen von vier Steinen -> **-1 Punkt**
Gelb hat vertikal schon einen von vier Steinen -> **-1 Punkt**

⇨ Wert des Spielfeldes: **-1 Punkt**

3.3.3 TurnCalculator Algorithmus

Mit den Möglichkeiten den Gewinner zu ermitteln und eine Spielsituation bewerten zu können, kann nun das „Gehirn“ der künstlichen Intelligenz programmiert werden: der TurnCalculator. Dieser hat zur Aufgabe aus der Fülle von möglichen Spielverläufen den vorteilhaftesten auszuwählen.

Genau wie der WinChecker und der GridEvaluator wurde auch der TurnCalculator so entwickelt, dass dieser unabhängig von der Ablauflogik des Spiels lauffähig ist. Es sind lediglich zwei Informationen nötig (der sich am Zug befindende Spieler und das Grid). Als Rückgabewert liefert der TurnCalculator dann die erfolgversprechendste Spalte zurück.

3.3.3.1 *MiniMax-Algorithmus*

Der MiniMax-Algorithmus dient dazu aus einer Auswahl von möglichen Zügen den vorteilhaftesten auszuwählen (Maximierung). Dabei wird jeweils davon ausgegangen, dass der Gegner immer so spielt, dass er ebenfalls den besten Zug für sich auswählt (Minimierung). Somit wird stets der höchstmögliche Gewinn, unabhängig von der Spielweise des Gegners erreicht.

In einer nichtoptimierten Variante werden alle möglichen Züge durchprobiert. In der untenstehenden Grafik ist veranschaulicht, wie aus der Gesamtheit der möglichen Züge ein Suchbaum erstellt wurde.

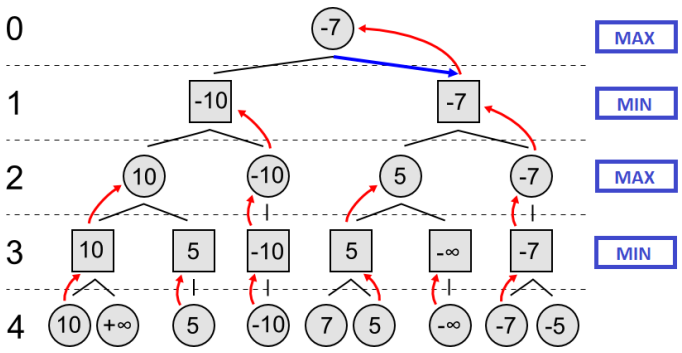


Abb. 36 Darstellung des MiniMax Baumes [Quelle: Wikipedia Minimax-Algorithmus]

Der sich am Zug befindende Spieler steht immer auf der Ebene 0. In diesem Beispiel hat er nun die Möglichkeit zwei verschiedene Züge zu machen (linker oder rechter Kindknoten). Danach ist der Gegner am Zug (Symbolisiert durch quadratische Knoten). Dieser hat jeweils wieder die Wahl zwischen 2 verschiedenen Knoten. Je nach Schwierigkeitsgrad wird der Baum bis zu einer gewissen Suchtiefe aufgebaut. Da immer abwechselndes Zugrecht herrscht, wechseln sich auf jeder Ebene Spieler und Gegner ab.

In den Blattknoten (Ebene 4) wird nun die erreichte Situation vom GridEvaluator bewertet. Danach werden die erhaltenen Werte nach oben weitergereicht. Bei allen ungeraden Ebenen ist der Gegner am Zug. Da dieser den aktuellen Spieler zu einem möglichst schlechten Zug zwingen möchte, gibt er immer den niedrigsten Wert nach oben weiter. Mit anderen Worten: Der Gegner spielt minimierend. Hingegen spielt der aktuelle Spieler so, dass er einen möglichst guten Zug findet. Deshalb gibt er immer den höchstmöglichen Wert weiter. Er spielt also maximierend. Sobald die Wurzel des Baumes erreicht ist, kann eine Entscheidung getroffen werden, welcher Zug der Beste ist.

Natürlich ist das oben dargestellte Beispiel eine stark vereinfachte Darstellung. Im spezifischen Fall unseres Vier Gewinnt Spieles ist der Baum wesentlich grösser. Solange noch keine Spalte des Spielfeldes voll ist, hat der Spieler immer sieben verschiedene Zugmöglichkeiten, sprich jeder Knoten hat sieben Kindknoten. Je mehr Züge voraussimuliert werden sollen, desto grösser wird die Suchtiefe und desto bessere Züge berechnet der Algorithmus. Dementsprechend schlecht skaliert der Suchbaum bei höheren Schwierigkeitsgraden. So ergeben sich z.B. schon bei einer Suchtiefe von 8 Zügen 7^8 (=5‘764‘801) Blattknoten die bewertet werden müssen. Würde man ein gesamtes Spiel simulieren wollen, müsste man 7^{42} = 311‘973‘482‘284‘542‘371‘301‘330‘321‘821‘976‘049 (rund 311 Dezillionen) Blattknoten bewerten. Da dies die Rechenkapazität des im Roboter vorhandenen Servers übersteigen würde, muss die Suchtiefe auch auf der schwersten Stufe limitiert werden.

3.3.3.2 *Optimierung: Alpha-Beta-Suche* [2]

Bei der Alpha-Beta-Suche handelt es sich um eine verbesserte Variante des MiniMax Algorithmus. Beim MiniMax Algorithmus wird immer der gesamte Baum analysiert. Dies beinhaltet unter anderem auch Blattknoten, welche das Endergebnis gar nicht beeinflussen. Das Prinzip der Alpha-Beta Suche lässt sich am besten am Beispiel eines einfachen Spieles erklären.

Abb. 37 Fiktive Kiste mit drei Geldscheinen

Man stelle sich vor, es stünden sich zwei Spieler (A und B) gegenüber. B besitze zwei Kisten mit jeweils drei Geldscheinen von unterschiedlichem Wert darin. Spieler A darf sich eine Kiste auswählen, aus welcher ihm B einen Geldschein abgeben muss. Da B ein sehr geiziger Mensch ist, gibt er A immer den Geldschein mit dem niedrigsten Wert.

Als erstes schaut A in die erste Kiste und findet dort Geldscheine mit den Werten 20$, 50$ und 100$ (siehe Abb. 37). Somit ist sich A bewusst, dass er bei der Wahl dieser Kiste 20$ bekäme. Er behält sich also diese 20$ im Hinterkopf.

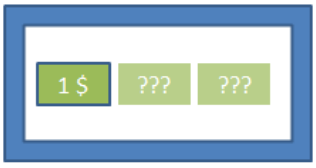


Abb. 38 Eine zweite Kiste

Danach nimmt er sich die zweite Kiste vor. Sofort erblickt er einen 1$-Geldschein (siehe Abb. 38). Die Untersuchung der anderen

[2] (Baier 2006)

beiden Scheine kann er sich also sparen, denn selbst wenn die anderen 1000$ oder mehr wären, er würde sowieso nur 1$ bekommen. Da er schon die 20$ aus Kiste 1 auf sicher hat, wählt er also in jedem Fall Kiste 1.

Mit dieser Methode wurden demnach nur vier der insgesamt sechs Elemente untersucht, ohne dass sich das Resultat dadurch verschlechtert hätte.

Genau dieses Prinzip lässt sich auch auf den Suchbaum unseres MiniMax Turncalculators anwenden. Durch die Einführung von zwei Werten (alpha und beta) müssen nicht mehr alle Blattknoten analysiert werden, was die Laufzeit der Zugsuche drastisch verkürzen kann. Der Alpha-Wert ist dabei das Ergebnis, das Spieler A mindestens erreichen wird, der Beta-Wert ist das Ergebnis, das Spieler B höchstens erreichen wird (Hierbei ist zu beachten, dass es Spieler B darum geht, ein möglichst niedriges Ergebnis zu erhalten, da er ja "minimierend" spielt!).

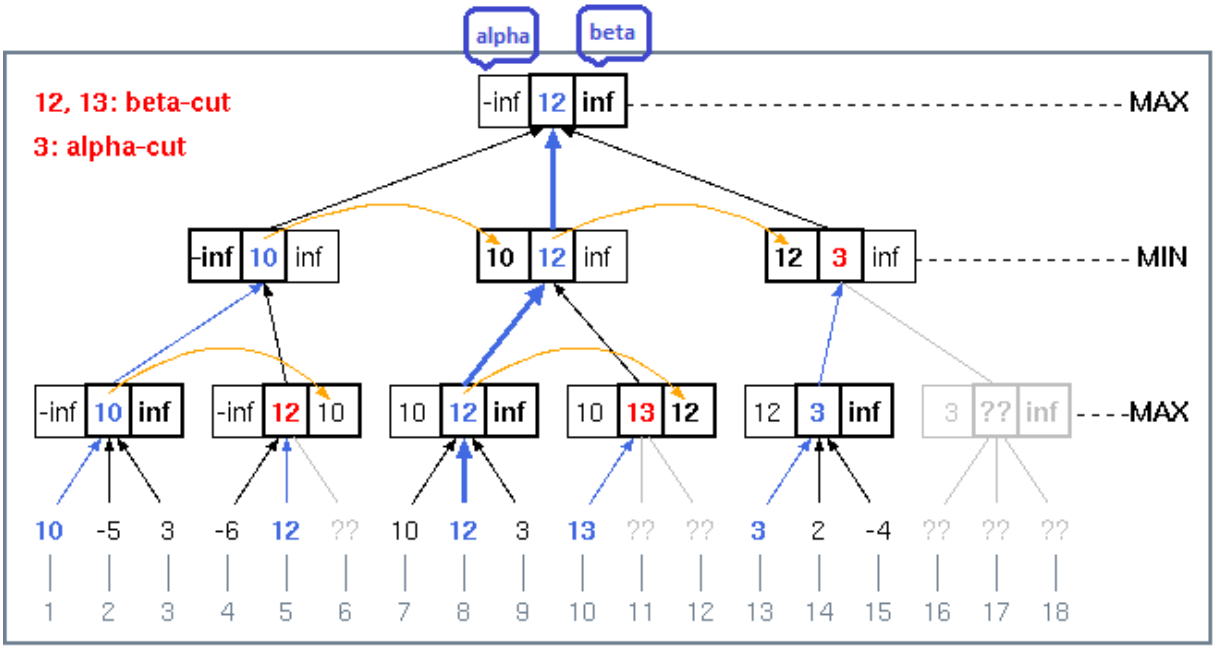


Abb. 39 Darstellung der Alpha Beta Suche [Quelle: Wikipedia Alpha-Beta-Suche]

Der obenstehenden Grafik ist zu entnehmen, dass die mit „??“ gekennzeichneten Blattknoten nicht ausgewertet werden müssen, da diese nicht zur Problemlösung beitragen. In diesem Beispiel könnten rund ein Drittel der Bewertungsfunktionen weggelassen werden, ohne das Resultat zu verschlechtern.

3.3.3.3 *Optimierung: Spielzug-Sortierung*

Eine weitere Optimierung kann durch eine Vorsortierung der Züge erreicht werden. Da die Alpha-Beta-Suche schneller arbeitet, je kleiner das Alpha-Beta-Fenster ist, kommt es auf die Reihenfolge der Züge an. Diejenigen Züge, welche vermutlich besser sind als andere, sollten deshalb zuerst untersucht werden. In unserem Fall werden die Züge so sortiert, dass die mittleren Spalten des Spielfeldes als erstes geprüft werden. Zwar ist diese Sortierung nicht in allen Fällen optimal, dennoch lassen sich durch diese Verbesserung markante Laufzeitverbesserungen messen.

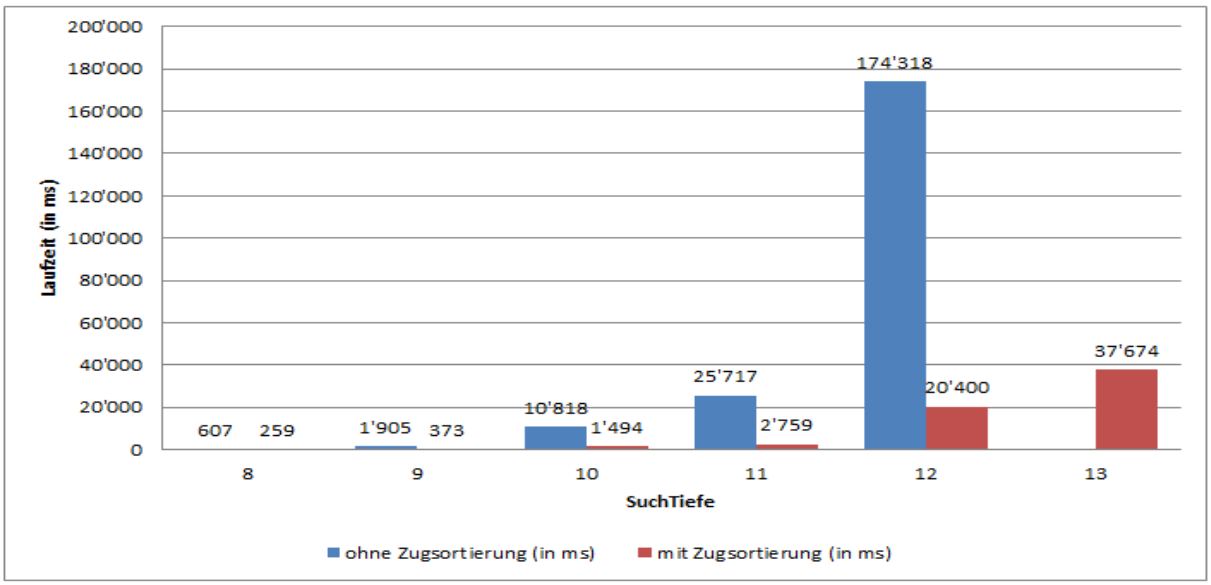


Abb. 40 Vergleich der Laufzeit mit und ohne Zugsortierung

Die in der Abbildung 40 dargestellten Werte beziehen sich alle auf ein leeres Spielfeld. Zur besseren Übersicht wurde der Wert für die Suchtiefe 13 ohne Zug-Sortierung weggelassen.

3.3.4 Weitere Algorithmus Optimierungsmöglichkeiten

Alle oben genannten Optimierungen wurden wie beschrieben implementiert. Jedoch liesse sich der ganze Algorithmus beinahe beliebig weiter optimieren. Allerdings würde dies den Rahmen einer Studienarbeit bei weitem sprengen. Dennoch seien hier die weiteren Möglichkeiten kurz beschrieben.

3.3.4.1 *Multithreading:*

Der Aufbau des MiniMax Suchbaumes könnte multi-threaded realisiert werden. Dabei könnten z.B. sieben Threads jeweils den Aufbau eines Subbaumes übernehmen.

Der Einfachheit halber wurde jedoch alles single-threaded implementiert. Schliesslich ist es auch fraglich, ob der MiniMax Algorithmus mit einer Alpha-Beta Optimierung tatsächlich von der Parallelisierung profitieren würde, da Alpha und Beta Werte Astübergreifend benötigt werden.

3.3.4.2 *Iterative Tiefensuche:*

Bei der iterativen Tiefensuche wird der Suchbaum zuerst mit einer geringen Suchtiefe aufgebaut. Bleibt man dabei unter einem vorgängig definierten Zeitlimit, wird die Suchtiefe erhöht. Da es sich beim MiniMax Algorithmus um eine Tiefensuche handelt, muss der gesamte Baum dabei erneut aufgebaut werden. Bei jeder weiteren Iteration können dann die Ergebnisse der jeweils vorherigen Runde genutzt werden, um z.B. die Zug-Sortierung zu optimieren oder Alpha und Beta mit einem Initialwert zu versehen.

3.3.4.3 *Erweiterte Bewertungsfunktion:*

Da der Zugfindungsalgorithmus im Kern unabhängig vom Spiel ist, hängt seine Stärke allein von der verwendeten Bewertungsfunktion ab. Der implementierte GridEvaluator weist in diesem Punkt noch einige Schwächen auf.

Eine erweiterte Variante könnte z.B. die Tatsache ausnutzen, dass Viererkombinationen auf den unteren Ebenen meist einfacher zu vervollständigen sind als auf den höheren, da man bei diesen noch darauf angewiesen ist, dass der Gegner hilft die Spalten aufzufüllen.

3.4 Testing

3.4.1 Analyse Tests

3.4.1.1 *Testen der Roboterschnittstelle*

Vor der Implementation der neuen Software wurde getestet, ob sich die Schnittstelle zum Roboter (gegeben aus alter SA) definitionsgemäss verhält. Des Weiteren war die Schnittstelle nicht gänzlich dokumentiert, es gab Fälle bei denen es unklar war wie sich der Roboter verhalten wird. Deshalb wurde entschieden ein Testprotokoll zu entwickeln, welches gleichzeitig als Schnittstellen Dokumentation für den Roboter verwendet werden kann (Abbildung 41).

Beschreibung	Vorbedingung	Test	Erwartetes Resultat	Tatsächliches Resultat	
Den aktuellen Status abfragen	Roboter ist Ready	Sende char 'A'	Roboter antwortet mit char '1' (Ready)	Roboter antwortet mit char '1' (Ready)	
Den aktuellen Status abfragen	Roboter ist Busy	Sende char 'A'	Roboter antwortet mit char '2' (Busy)	Roboter antwortet mit char '2' (Busy)	
Den aktuellen Status abfragen	Roboter hat Error	Sende char 'A'	Roboter antwortet mit char '3' (Error)	Roboter antwortet mit char '3' (Error)	
Roboter initialisieren		Sende char 'B'	Nichts	Nichts	1)
Neues Spiel starten		Sende char 'C'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	2)
Unentschieden signalisieren		Sende char 'D'	Nichts	Nichts	3)
Rot gewinnt		Sende char 'E'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	4)
Gelb gewinnt		Sende char 'F'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 1		Sende char 'G '	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	5)
Roter Stein Feld 2		Sende char 'H'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 3		Sende char 'I'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 4		Sende char 'J'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 5		Sende char 'K'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 6		Sende char 'L'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Roter Stein Feld 7		Sende char 'M'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 1		Sende char 'N'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 2		Sende char 'O'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 3		Sende char 'P'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 4		Sende char 'Q'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 5		Sende char 'R'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 6		Sende char 'S'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Gelber Stein Feld 7		Sende char 'T'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Schalte LED1 ein		Sende char 'U'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	6)
Schalte LEDs 2-5 ein		Sende chars 'V' - 'Z'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	7)
Schalte LEDs 6-31 ein		Sende chars 'a' - 'z'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	
Schalte LEDs 32-42 ein		Sende chars '1', '2', '3'...'0'	Roboter antwortet mit char '4' (Received)	Roboter antwortet mit char '4' (Received)	

Generell gilt
Jedes empfangene Zeichen wird vom Robo mit '4' (RECEIVED) beantwortet.
Zeichen können nur empfangen werden wenn Robo nicht BUSY ist! (Ausnahme, Zeichen 'A' kann immer empfangen werden.)

Folgende Zustände lassen sich ableiten nach senden eines dieser Zeichen
Rot = Robo BUSY. Diese Zeichen werden mit RECEIVED beantwortet sofern Robo nicht BUSY, danach ist Robo für längeren Zeitraum BUSY
Hellgrün = Robo READY. Diese Zeichen werden mit RECEIVED beantwortet sofern Robo nicht BUSY, Robo bleibt immer READY
Dunkelgrün = Robo READY. Diese Zeichen werden immer mit RECEIVED beantwortet egal ob Robo BUSY, Robo bleibt immer READY
Gelb = Deaktivierte Funktion

Kommentar
1) Roboter macht nichts (momentan deaktiviert)
2) Steine fallen raus und werden wieder gesammelt.
3) Roboter macht nichts (momentan deaktiviert)
4) Es Passiert nichts, bis 4 LED positionen empfangen werden.
5) Wird ein G und dann ein M gesendet, so wird M genommen, sofern der Einwurfarm mit dem stein noch nicht unterwegs ist.
6) Zuerst Gewinner melden, sonst leuchtet nichts. Es leuchtet immer erst nach dem 4. LED. Roboter resetet automatisch nach ca. 10 Sekunden.

Abb. 41 Testprotokoll der Roboterschnittstelle

Chars for LEDs						
4	5	6	7	8	9	0
w	x	y	z	1	2	3
p	q	r	s	t	u	v
i	j	k	l	m	n	o
b	c	d	e	f	g	h
U	V	W	X	Y	Z	a

Y	X/Y Coords						
05	15	25	35	45	55	65	
04	14	24	34	44	54	64	
03	13	23	33	43	53	63	
02	12	22	32	42	52	62	
01	11	21	31	41	51	61	
00	10	20	30	40	50	60	X

Abb. 42 Zeichen der LEDs sowie deren Koordinaten

Das rechte Raster in Abbildung 42 zeigt ein kartesisches Koordinatensystem mit den Vier Gewinnt Spielfeld Dimensionen von der Breite sieben und der Höhe sechs. Das linke Raster zeigt zu jeder Koordinate des rechten Rasters das Zeichen, welches an den Roboter gesendet werden muss.

Sollen die LEDs ganz links oben {(0/5), (1/5), (2/5), (3/5)} angesteuert werden, müsste zuerst die Gewinnerfarbe (siehe Abbildung 41) und danach die Zeichen ‘4‘, ‘5‘, ‘6‘ und ‘7‘ an den Roboter gesendet werden.

3.4.2 Unit Tests

Während der ganzen Implementationsphase wurden fortlaufend neue Unit Tests geschrieben. Am Ende sind es insgesamt 118 Unit Tests.

3.4.2.1 *Einführung*

Da dieses Projekt über drei Tiers hinweg läuft, siehe Kapitel 4.1.1, und ausserdem etliche verschiedene Technologien eingesetzt werden war das Testen einiger Packages beinahe unmöglich oder hätte mit Faking & Mocking einen Aufwand bedeutet, der im Umfang dieser SA nicht tragbar gewesen wäre. Des Weiteren wurden in dieser Arbeit etablierte Frameworks und Technologien verwendet, die bereits durch dessen Hersteller getestet wurden.

3.4.2.2 *Nicht getestete Packages*

ch.hsr.connectfourrobo.controller

Dieses Package enthält nur den sogenannten Webcontroller. Dieser bestimmt mit Hilfe der Spring Annotationen @Controller und @RequestMapping die Anlaufstelle für die entsprechenden URL-Aufrufe und ruft auf dem Service Layer die benötigten Methoden auf. Es ist deshalb nicht nötig dieses Package zu testen da es nur als Zuweisungspackage fungiert.

ch.hsr.connectfourrobo.test

Das Test Package wird nicht getestet, da dieses die eigentlichen Testfunktionen enthält.

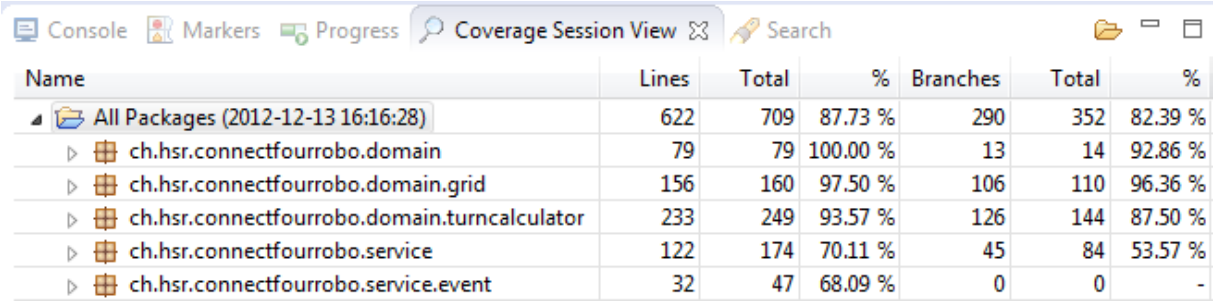
ch.hsr.connectfourrobo.domain.robot

Dieses Package stellt die Kommunikation zum Roboter über RS232 dar. Da sich das Mocken der seriellen Schnittstelle über verschiedene Betriebssysteme hinweg als enorm aufwändig und schwierig herausstellte, wurde dieser Teil weggelassen. Ausserdem wird mit einem Test des Package zur seriellen Kommunikation keinen Gewinn erzielt, wenn das Package aufgrund verschiedener OS Implementationen gemockt werden muss. Damit dieser Teil trotzdem nicht ungetestet bleibt, wurde ein RoboterSimulator entwickelt, der manuell gestartet und somit auch für ein nächstes Projekt wiederverwendet werden kann.

ch.hsr.connectfourrobo.util

Dieses Package verwendet nur Java Grundfunktionen oder 3rd-Libraries welche bereits durch dessen Hersteller getestet wurden.

3.4.2.3 *Testabdeckung*



Name	Lines	Total	%	Branches	Total	%
All Packages (2012-12-13 16:16:28)	622	709	87.73 %	290	352	82.39 %
ch.hsr.connectfourrobo.domain	79	79	100.00 %	13	14	92.86 %
ch.hsr.connectfourrobo.domain.grid	156	160	97.50 %	106	110	96.36 %
ch.hsr.connectfourrobo.domain.turncalculator	233	249	93.57 %	126	144	87.50 %
ch.hsr.connectfourrobo.service	122	174	70.11 %	45	84	53.57 %
ch.hsr.connectfourrobo.service.event	32	47	68.09 %	0	0	-

Abb. 43 Testabdeckung

Die Domain, welche den grössten Teil der Logik des Spiels enthält, wurde sehr ausgiebig getestet. Sofern die oben begründeten weggelassenen Packages nicht in die Testabdeckung einbezogen werden, wurde eine Testabdeckung von 88% erreicht. Da die restlichen 12% fast nur Setter-/Getter-Methoden sind, wurde darauf verzichtet Tests für diese zu schreiben.

3.4.3 System Tests

3.4.3.1 *Einführung*

Um die Software auch ohne Roboter testen zu können, wurde eine Konsolenvariante des Spieles implementiert, welche dazu genutzt werden kann ein Spiel zu simulieren. Dieses Konsolenspiel nutzt einen eigens zu testzwecken programmierten Robotersimulator, der mechanische Arbeit simuliert und über die gleichen Schnittstellen wie der physikalische Roboter kommuniziert.

Da der Robotersimulator unabhängig vom Rest des Projekts läuft, kann dieser als Referenzimplementation des Roboters für zukünftige Projekte benutzt werden.

Nach der Implementationsphase wurden am physikalischen Roboter ausführliche Systemtests durchgeführt. Durch eine Demonstration in der HSR Cafeteria wurde die neue Software schliesslich einem realen Praxistest unterzogen.

3.4.3.2 *Testen der Use Cases*

Aufgrund der Use Cases lassen sich folgende relevante System Tests ableiten.

Nr.	Testbeschreibung	Resultat
1	Zwei Personen möchten gegeneinander ein Spiel starten. Beide öffnen je auf einem WIFI-Gerät einen Browser, begeben sich in die Spiellobby und starten ein Spiel.	Erfolgreich
1.1	Die Spieler spielen gegeneinander bis ein Sieger feststeht. Der Roboter führt die Züge des Spielfelds auf dem Roboter Grid durch.	Erfolgreich
1.1.0	Der Verlierer erhält die Meldung, dass er verloren hat.	Erfolgreich
1.1.1	Der Sieger erhält die Meldung, dass er gewonnen hat.	Erfolgreich
1.1.2	Der Roboter zeigt die Gewinnpositionen auf dem Spielfeld mit der Gewinnerfarbe an.	Erfolgreich
1.1.3	Beide Spieler können sich nach dem Spiel in die Lobby zurück begeben.	Erfolgreich
1.2	Die Spieler spielen so, dass sich ein Unentschieden einstellt. Der Roboter führt die Züge des Spielfelds auf dem Roboter Grid durch.	Erfolgreich

1.2.0	Beide Spieler erhalten die Meldung des Unentschieden.	Erfolgreich
1.2.1	Der Roboter schaltet keine LEDs an.	Erfolgreich
1.2.2	Beide Spieler können sich nach dem Spiel in die Lobby zurück begeben.	Erfolgreich
2	Eine Person startet auf einem WIFI-Gerät in einem Browser ein Spiel gegen eine künstliche Intelligenz.	Erfolgreich
2.1	Der künstliche Spieler gewinnt das Spiel. Der menschliche Spieler erhält die Verlierermeldung.	Erfolgreich
2.1.0	Der menschliche Spieler kann sich in die Lobby zurückbegeben.	Erfolgreich
2.2	Der menschliche Spieler gewinnt das Spiel. Der menschliche Spieler erhält die Siegermeldung.	Erfolgreich
2.2.0	Der menschliche Spieler kann sich in die Lobby zurückbegeben.	Erfolgreich
3	Es wird ein Spiel gegen einen künstlichen Spieler gestartet. Danach wird in einem anderen Browserfenster auf die Visualisierung zugegriffen.	Erfolgreich
3.1	Nach jedem Zug des künstlichen Gegners wird ein neuer Baum dargestellt.	Erfolgreich
4	Zwei Personen möchten gegeneinander ein Spiel starten. Beide öffnen auf einem WIFI-Gerät einen Browser, begeben sich in die Spiellobby und starten ein Spiel.	Erfolgreich
4.1	Spieler1 verlässt das Spiel und gelangt zurück in die Lobby.	Erfolgreich
4.2	Spieler2 sieht die Meldung, dass Spieler1 das Spiel verlassen hat und kann in die Lobby zurückkehren.	Erfolgreich

3.4.4 Usability Tests

Die Usability wurde wie folgt getestet:

- Innerhalb des Studienarbeitsraumes wurden verschiedene Studenten dazu angehalten das Spiel zu spielen und somit zu testen. Das dabei erhaltene Feedback ist vor zu in die Implementation der Lösung eingeflossen.
- Die Benutzeroberfläche wurde auf verschiedenen WIFI-Geräten und Browsern erfolgreich getestet. Geräte die nicht unterstützt werden, erhalten eine entsprechende Meldung beim Aufruf des Spiels.
- Zudem wurde das GUI direkt am Roboter mit dem Roboterbauer getestet. Da dieser einerseits die alte Software sehr gut kennt und andererseits viele Erfahrungen mit Ausstellungsbesuchern sammeln konnte, hat er uns sehr gute Tipps zur Verbesserung des GUIs weitergeben können. Ein hilfreicher Hinweis war unter anderem, dass die Benutzer bei einem vorzeitigen Spielabbruch aufgrund Inaktivität eine frühzeitige Rückmeldung erhalten sollen. Diese Funktion wurde danach besser umgesetzt und erneut getestet.
- Das GUI und die Visualisierung wurde der Abteilungsleiterin der Abteilung Maschinenbau sowie dem Roboterbauer am 11.12.2012 im Labor am Roboter demonstriert. Während der Demonstration wurde der Verbesserungsvorschlag geäussert die Spielsteine im Baum mit der Farbe des jeweiligen Spielers hervorzuheben. Dies wurde entsprechend umgesetzt und am 14.12.2012 nochmals an der Live-Demo präsentiert.
- Am 14.12.2012 wurde während einer 1,5 stündigen Live-Demo das Gesamtprojekt mit Einbezug des Roboters an der HSR präsentiert. Die Live-Demo die auch als Abschlusstest genutzt wurde, hat gezeigt, dass den Testpersonen der Umgang mit der neuen Software leicht gefallen ist.

3.5 Ergebnisse / Umsetzung

3.5.1 Erreicht

Visualisierung implementiert

Es wurde eine neue Funktionalität in die Software integriert: Die Visualisierung des Spielalgorithmus. So wird der Roboter auch für die potenziellen Informatikstudenten interessanter, da ein Einblick in den Softwareteil und somit die Arbeit eines Informatikers gewährt wird.

Software vereinfacht

Mit dem Einsatz von neuen Technologien wie dem Spring MVC-Framework konnte die Software weitgehend verbessert werden. Dadurch wird die Wartung und Erweiterung der Software in Zukunft stark vereinfacht.

Fehlerbehandlung verbessert

Die neue Software ist gesamthaft fehlerresistenter als die alte. So musste früher bei einem Roboterfehler auch der Serverprozess neu gestartet werden. In der neuen Version regeneriert sich die Software von selbst und kehrt in einen konsistenten Zustand zurück. Es kann direkt weitergespielt werden, sobald das Problem am Roboter behoben wurde.

Schwierigkeitsstufen angepasst

Bis anhin war es selbst auf der einfachsten Stufe schwierig gegen den Roboter zu gewinnen, da dieser im Gegensatz zu einem menschlichen Gegner nie einen Fehler gemacht hat. Neu wurde die Möglichkeit implementiert zufällige Züge einzustreuen. Ausserdem lassen sich die Schwierigkeitsstufen nun in einer Konfigurationsdatei anpassen, ohne die Software neu kompilieren zu müssen.

Video erstellt

Um Werbung für ein Studium an der HSR zu machen, wurde ein Video produziert, welches einen Einblick in das Projekt liefert. Es bietet sowohl einen Einblick in die Arbeitsweise als auch eine Demonstration des Endproduktes.

3.5.2 Nicht erreicht

Demo-Modus (Persistenz der Spiele)

Entgegen der ursprünglichen Aufgabenstellung wurden die gespielten Spiele nicht gespeichert da für den Auftraggeber eine bessere Lösung gefunden wurde.

Nicht unterstützte Geräte und Browser

Leider wird der Native Browser von Android Versionen unter und mit 2.3.5 aufgrund von Inkompatibilität mit dem Atmosphere Framework bzw. SpringSource nicht unterstützt. Es könnte jedoch sein, dass mit kommenden Versionen von SpringSource oder Atmosphere die oben genannten Android Versionen unterstützt werden. Momentan wird dem Benutzer empfohlen einen alternativen Browser zu verwenden.

Des Weiteren werden iOS-Geräte mit der Version 5.1 ebenfalls nicht unterstützt, da Apple in dieser Version eine veraltete Spezifikation der Websocket Technologie implementiert hat. Ein

Workaround wäre gemäss Jeanfrancois Arcands [3] Artikel die Webapplikation unter Glassfish anstatt Tomcat zu starten. Diese Problematik wurde leider erst gegen Ende des Projekts ersichtlich, da die zum Testen zur Verfügung gestellten Geräte entweder auf der neusten iOS Version 6 oder auf einer älteren Version unter 5.1 liefen. Des Weiteren wurde die Software unter Tomcat entwickelt und die Portierung auf Glassfish konnte nicht innert nützlicher Frist reibungslos umgesetzt werden.

3.6 Schlussfolgerungen

In diesem Abschnitt wird jeweils zuerst kurz auf die Kernprobleme der Arbeit eingegangen und dann erläutert in wie fern dies gelöst werden konnte. Am Ende des Kapitels wird ein Gesamtfazit gezogen inklusive Ausblick was noch verbessert werden könnte.

3.6.1 Stabilität

Problem

Aufgrund schlechter Programmierung sind Nebenläufigkeiten entstanden, welche vermehrt zum Absturz der alten Software Lösung führten. Seitens des IFS wurde versucht kritische Funktionen mit Java-Synchronized abzusichern. Dieser Workaround hat zwar zu einer Minimierung der zufälligen Abstürze, aber nicht zur Beseitigung deren geführt. Da das Gesamtprojekt als Ausstellungsobjekt an Infotagen präsentiert wird, sind Ausfälle der Software nicht tragbar.

Ergebnis

In der neuen Lösung wurde deshalb bewusst so wenig Asynchronität verwendet wie möglich. Wie im Kapitel Design zum Sequenzdiagramm detailliert erläutert wird, wurden aufgrund der HTTP Request bis zum Service Layer Asynchrone Methoden Aufrufe, welche mittels Java-Synchronized gesichert sind, verwendet. Unterhalb des Service Layers, also in der ganzen Domain sowie der Kommunikation zum Roboter, verlaufen die Methodenaufrufe synchron. Somit konnte die Komplexität minimiert und die Probleme aufgrund von Nebenläufigkeiten gelöst werden. Des Weiteren kann der Controller Layer bzw. die ganze Frontend Lösung unproblematisch ausgetauscht werden.

3.6.2 Wartung

Problem

Die Assistenten des IFS hätten das alte Projekt warten müssen. Die Assistenten haben sich intensiv mit dem Code der alten Lösung auseinander gesetzt, sind aber zur Erkenntnis gekommen, dass der Code ohne grösseren Aufwand nicht optimier- und ausbaubar war. Anleitung zur Installation bzw. Wartung der alten Lösungen seien nicht vorhanden oder nicht detailliert genug gewesen.

Lösung

Die Stakeholder, in diesem Fall die Assistenten, wurden von Beginn an in die Erarbeitung des Projekts miteinbezogen, wobei unteranderem auf die Probleme der alten Lösung aufmerksam gemacht wurde. Des Weiteren konnten dadurch gewünschte Technologien der Assistenten wie beispielsweise das Spring MVC Framework direkt in der neuen Lösung umgesetzt werden. Ausserdem wurde den Assistenten nach Erarbeitung einer ersten stabilen Version der neuen Software, der Code sowie die zu diesem Zeitpunkt bestehende Dokumentation zum Review überge-

[3] (Arcand 2012)

ben. Nach dem Review wurden sämtliche bemängelte Aspekte überarbeitet. Mit dem Einbezug der Assistenten konnten gewünschte Änderungen direkt umgesetzt und die Betreuung sowie Wartung des Projekts vereinfacht werden.

3.6.3 Schwierigkeitsgrad

Problem

Von allen Seiten, also dem Auftraggeber, der Assistenten des IFS, dem Roboterbauer sowie den jeweiligen Roboterbetreuern an den Infotagen wurde bemängelt, dass sich der Schwierigkeitsgrad des Spiels gegen den künstlichen Spieler nicht wie erwartet verändern lasse. Das Spiel war somit generell zu schwierig und für die meisten Infotag Besucher schnell unattraktiv, da sich ohne fehlende Gewinnmotivation kein Game Flow einstellte. Die Tiefe des Algorithmus, also die Anzahl Züge welche vorausberechnet werden, wurde verringert. Diese Änderung erbrachte jedoch nicht den gewünschten Erfolg. Denn der künstliche Gegner spielte dann so einfach, dass der Spieler beinahe jedes Spiel gewonnen hat.

Lösung

Die Umsetzung eines ersten Prototyps des Algorithmus wurde bereits zu Beginn des Projekts angestrebt. Danach wurde ein einfaches, primitives Konsolenspiel programmiert mit welchem die Probleme beim Spielen gegen den Algorithmus schnell erkannt wurden. Denn dabei wurde festgestellt, dass der künstliche Gegenspieler im Vergleich zu einem Menschen keine Fehler macht. Da über die Suchtiefe die Intelligenz des Algorithmus beeinträchtigt werden kann, wurde entschieden die drei Schwierigkeitsstufen EASY, MEDIUM und HARD einzuführen. Auf EASY beträgt die Suchtiefe lediglich zwei, auf MEDIUM fünf und auf HARD werden acht Züge voraus simuliert. Des Weiteren wurde der Stufe EASY ein Parameter hinzugefügt, mit Hilfe dessen die Wahrscheinlichkeit definiert wird, ob der nächste Spielzug ein unvorteilhafter Zug sein soll. Der Parameter zur Angabe der Anzahl Random Züge lässt sich im Konfigurationsfile in Prozent angeben. Als guter Default Wert wurde 0.33 ermittelt, was dann also bedeutet, dass jeder Zug mit einer Wahrscheinlichkeit von 33% ein schlechter Zug ist. Im Live-Test hat sich gezeigt, dass es den Spielern mit der Stufe EASY nun möglich war gegen die künstliche Intelligenz zu gewinnen. Der Anreiz gegen den Roboter mit einen höheren Schwierigkeitsgrad zu spielen, wurde somit geschaffen. So ist es z.B. für Spieler mit einem durchdachten System nun interessant herauszufinden, ob ihr System auch gegen die neue Implementation des MiniMax Algorithmus bestehen kann.

3.6.4 Visualisierung

Problem

Das Ziel des Auftraggebers dieses Projekts ist es, den Spielern mittels Roboter den Informatik-Studiengang an der HSR näher zu bringen und somit schmackhaft zu machen. Bis anhin wurden die Besucher aufgrund des spektakulären Roboters aber nur auf den Maschinenbaustudiengang aufmerksam. Der Softwareteil bzw. der Algorithmus und somit der spannende Teil der Informatik blieb dem Besucher des Infotags weitgehend verborgen.

Lösung

Zwar hatte der Auftraggeber keine konkrete Vorstellung der Darstellung zur Visualisierung des Algorithmus, wünschte sich aber eine solche. Alleine die Erläuterung eines Algorithmus im Allgemeinen gestaltet sich als sehr schwierig. Diesen dann aber an einer Messe unterschiedlichsten

Besuchern innert kürzester Zeit zu erläutern ist in Anbetracht des Zeitrahmens der Studienarbeit aber ein Ding der Unmöglichkeit. Eine komplette Erklärung des Algorithmus in einer Visualisierung wird somit ausgeschlossen. Nach der Vorstellung vier verschiedener Lösungsvarianten zur Visualisierung des Algorithmus beziehungsweise das hervorheben des Informatikteils der Arbeit, wurde aufgrund der Präferenz des Auftraggebers eine Baumvisualisierung angestrebt. Mittels der Baumvisualisierung kann nachvollzogen werden, weshalb sich der künstliche Gegner für seinen letzten Zug entschieden hat. Auch die Abteilung Maschinenbau hat diese Idee als interessant erachtet und konnte überzeugt werden einen weiteren Bildschirm zur Visualisierung der Züge des künstlichen Gegners in der Nähe des Roboters anzubringen. Beim Live-Test hat sich herausgestellt, dass interessierte Spieler aufgrund der Visualisierung ein Gespräch in Richtung Informatik mit dem Roboterbetreuer suchten. Somit wird dem Informatikteil der Arbeit durch den zusätzlichen Bildschirm sowie der darauf laufenden Visualisierung nun viel mehr Aufmerksamkeit geschenkt.

3.6.5 Erweiterbarkeit & Austauschbarkeit

Problem

Die Idee des Auftraggebers sowie der Abteilung Maschinenbau ist es, dass der Roboter fortlaufend weiterentwickelt bzw. mit neuen Funktionalitäten erweitert werden kann. Die Assistenten des IFS hatten den Auftrag abzuklären, ob sich der Code verbessern lässt und ob mit dem bestehenden Code mit wenig Aufwand Erweiterungen eingebaut werden können. Die Assistenten sind zum Entschluss gekommen, dass sich in diesem Zustand des Codes kaum Erweiterungen einbauen liessen. Ausserdem wäre es aufgrund der Komplexität kaum möglich gewesen teile der Software auszutauschen.

Lösung

In Anbetracht dieser erkannten Problemstellung wurde in diesem Projekt angestrebt das Design sowie den Code möglichst verständlich und austauschbar zu gestalten. Es wurde gewissenhaft darauf geachtet, dass die Abhängigkeit zwischen den verschiedenen Schichten möglichst gering gehalten und auf keinen Fall von tieferen Layern auf obere Layer zugegriffen wird. Um ein konkretes Beispiel hierfür zu nennen, ist es also ohne Änderung des Codes möglich anstelle des Web-GUIs einen native Client zu implementieren. Somit ist garantiert, dass die verschiedenen Layer problemlos überarbeitet oder gar ausgetauscht werden können.

Ausserdem wurde die Qualität des Codes durch Reviews innerhalb der Projektbeteiligten sowie auch durch ein Review der IFS Assistenten sichergestellt. Des Weiteren wurden die geläufigsten Tools zur Analyse des Codestils sowie zur Prüfung von Fehlern im Code verwendet. Jede Klasse wurde mit Javadoc dokumentiert, womit sich zusätzlich mit wenigen Klicks eine Gesamtdokumentation des Codes generieren lässt.

Bei Anpassungen welche den Ablauf bzw. die Kommunikation des Roboters betreffen, kann auf einen eigenständigen speziell erstellten Robotersimulator zurückgegriffen werden. Somit wird die Entwicklung bzw. Anpassung des Codes bereits enorm vereinfacht. Durch die grosse Menge an Unit-Tests im Bereich der Domain wird sichergestellt, dass fehlerhafte Änderungen sofort bemerkt werden. Somit ist eine gute Ausganglage für zukünftige Erweiterungen gegeben.

3.6.6 Gesamtfazit

Abschliessend lässt sich sagen, dass das Redesign der Roboter-Software erfolgreich war. Der Roboter kann nun besser präsentiert werden, da die Stabilität enorm verbessert werden konnte. Bei Auftreten von Fehlern werden diese kontrolliert abgehandelt, so dass es zwar noch zu Spielunterbrüchen kommen kann, sicherlich aber nicht das ganze System davon betroffen ist. Somit sind Neustarts des Servers nun nicht mehr nötig und innert akzeptabler Zeit regeneriert sich das System im Normalfall von selbst. Es besteht somit eine solide Basis für allfällige Weiterentwicklung seitens der Abteilung Maschinebau, wie auch seitens der Abteilung Informatik. Die gewünschten Verbesserungen wie, bessere Stabilität der Software, adäquate Schwierigkeitsstufen und somit besser Game-Flow, mehr Aufmerksamkeit für den Informatikteil der Arbeit durch eine Visualisierung des MiniMax-Algorithmus, konnten somit erreicht und im Abschlusstest direkt mit Probanden getestet werden.

3.6.6.1 *Ausblick*

In absehbarer Zeit könnten folgende Punkte umgesetzt werden.

Momentan unterstützt das Spring Framework noch keine WebSockets, mit dem zusätzlichen Athmosphere Framework lassen sich diese aber trotzdem nutzen. In der nächsten Version des Spring Frameworks werden WebSockets unterstützt und somit bietet sich ein Umbau der Software mit nativer Implementation sicherlich an. Dadurch könnte der Code vereinfacht werden und die Handhabung der Clients über den Browser könnte sicherlich nochmals verbessert werden. Des Weiteren wäre es eine Überlegung wert vom Tomcat Webserver auf Glassfish umzusteigen, da WebSockets erfahrungsgemäss dort etwas besser gehandhabt werden. Theoretisch müssten .War-Files auf beiden Containern laufen, praktisch ist dies bei unserer für Tomcat entwickelten Lösung nicht der Fall, eine Anpassung würde aber sicherlich nur wenige Stunden Aufwand bedeuten.

Für die Anpassung des Config Files muss in der jetzigen Implementation das .War-File entpackt werden, das Config File angepasst werden und danach wieder in ein .War-File gezippt werden, danach muss das War-File redeployed werden. Dies gilt als normale Praxis für .War-Files, jedoch könnte man dies elegant umgehen. Es könnte ein Admin-Panel bereitgestellt werden, auf welchem man direkt die Parameter anpassen könnte, und danach per Knopfdruck den Server neustarten könnte.

Die Visualisierung des Algorithmus der künstlichen Spielzüge ist momentan darauf ausgelegt, dass in erster Linie Zuschauer angezogen werden sollen und ein Betreuer dann bei Bedarf Informationen zum Minimax-Algorithmus geben kann. Falls der Roboter zu einem späteren Zeitpunkt Autonom also ohne Betreuer laufen soll, müsste man die Visualisierung sicherlich anpassen. Es müsste enormer Aufwand investiert werden in die Visualisierung sowie ein neues Erklärungskonzept erstellt werden, so dass der Minimax-Algorithmus ohne Erläuterung eines Betreuers verstanden werden kann. Die Austauschbarkeit der Visualisierung sollte jedoch problemlos möglich sein, da bei der Implementation extra darauf geachtet wurde, dass jegliche Abläufe auch ohne die bestehende Visualisierung funktioniert. Somit hätte man die Möglichkeit die bestehende Visualisierung zu erweitern, oder diese ganz zu entfernen und ein ganz neues Konzept einzuführen.

Im Bereich der Benutzerinteraktion sowie des GUIs für den Benutzer hat sich gezeigt, dass sich problemlos eine neue Arbeit erstellen liesse. Denn in der momentanen Lösung wurden bewusst nur die nötigsten Features umgesetzt und der Fokus lag absolut auf der Kompatibilität für all die

verschiedenen Geräte wie Beispielsweise Laptops, Tablets und Smartphones sowie deren verschiedene Browser. Die Konzentrierung auf diese Aufgabe alleine hat sich bereits als trickreich und schwierig erwiesen, da die Webtechnologien sich ständig ändern und sich leider noch immer sehr schlecht an Standards halten. Deshalb könnte man in einer allein auf die Benutzerinteraktion und das GUI für den Benutzer designierten Arbeit sicherlich noch viele sinnvolle Features einbringen.

4 Glossar

Begriff	Erklärung
Atmosphere	WebSockets Library für Java
Coin	Ein Spielstein der eingeworfen werden kann
ConnectFour	Vier Gewinnt
CrossSiteScripting	Bezeichnet das Ausnutzen einer Sicherheitslücke in Webanwendungen
GitHub	Freies Versionsverwaltungssystems
Grid	Das Vier Gewinnt Spielbrett
GridSpace	Platzhalter für einen Coin im Grid
Human Player	Repräsentation eines menschlichen Spielers
IFS	Institut für Software
Java EE	Java Enterprise Edition
JSON	JavaScript Object Notation
KI	Künstliche Intelligenz
MiniMax	Verwendeter Algorithmus zur Zugbestimmung
Responsive Design	Ein sich auf die Auflösung anpassendes Design
SpringSource	SpringSource Framework, baut auf Java EE auf
STS	Spring Tool Suite (IDE)
Virtual Player	Repräsentation eines virtuellen Spielers
WebSockets	Bidirektionale Verbindung zwischen Webanwendung und Server
WinChecker	Algorithmus zur Bestimmung eines Siegers im Grid

5 Literaturverzeichnis

Arcand, Jeanfrancois. *Safari's WebSocket implementation and Java: Problematic!* 16. Mai 2012. http://jfarcand.wordpress.com/2012/05/16/safaris-websocket-implementation-and-java-problematic/.

Baier, Hendrik. „Der Alpha-Beta-Algorithmus und Erweiterungen bei Vier Gewinnt ." Bachelorarbeit, Offenbach, 2006.

Macdonald, Craig. *Fluid Squares V2.* 18. Juli 2011. http://dinosaurswithlaserz.com/2011/07/18/fluid-squares-v2/.

Wikipedia. *Lines of Code.* 17. Oktober 2012. http://de.wikipedia.org/wiki/Lines_of_Code.