Programmieren & Datenstrukturen

Dokumentation Projekt "Vier gewinnt"

P. Sollberger

Versionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Datum | Autor | Bemerkungen |
| 1.0 | 13.10.16 | Dozierendenteam | Initiale Version |

**Inhalt**

1. Einleitung 2

1.1. Begriffe, Abkürzungen 2

2. Anforderungen 3

2.1. Systemübersicht 3

2.2. Anwendungsfall 4

2.3. Tipps zur Vorgehensweise 6

2.4. Environment 6

3. Systemspezifikation 7

3.1. Bausteinsichten 7

3.1.1. Systemübersicht 7

3.1.2. Klassendiagramme 7

3.1.3. MVC 7

3.2. Laufzeitsichten 8

3.2.1. Zustandsautomat 8

3.2.2. Sequenzdiagramme 8

3.3. Verteilungssicht 9

3.4. Datensicht 9

3.5. Netzwerkprotokoll 9

4. Erweiterungsmöglichkeiten 9

Literaturverzeichnis 9

# Einleitung

Geschätzte Studierende

Im letzten Teil des Semesters bearbeiten Sie in einem kleinen Team ein Softwareprojekt. Mit diesem Projekt sind folgende Ziele verbunden:

* Sie wenden im Unterricht gelernte Konzepte der Sprache Java in einem grösseren Kontext an.
* Sie wiederholen wesentliche Elemente der Programmiersprache Java.
* Sie implementieren eine Softwarelösung im Team.
* Sie können Klassendiagramme lesen und interpretieren.
* Sie können Zustandsdiagramme lesen und interpretieren.
* Sie können Source-Code mit Hilfe von Klassendiagrammen und Sequenzdiagramme dokumentieren.
* Sie können in einem grösseren Programm die Übersicht wahren.

Das gesamte Projekt ist als Lernprojekt zu verstehen, bei dem Sie Schritt für Schritt Kenntnisse erwerben und anwenden.

Die Dozierenden und Assistierenden begleiten Sie während des Projekts und ermöglichen Ihnen, wesentliche Erfahrungen zu reflektieren. Zur Unterstützung dieses Prozesses erstellen Sie jede Woche für die Dozierenden einen kurzen Projekt-Statusrapport mit folgendem Inhalt:

* Welche Arbeiten wurden in der letzten Woche ausgeführt. Was hat gut geklappt, wo hatten oder haben Sie Probleme?
* Welche Tätigkeiten sind für die nächste Woche vorgesehen?
* Welche Knackpunkte (Herausforderungen oder Risiken) bestehen noch? Was gedenken Sie dagegen zu unternehmen?

Eine Vorlage für diesen Projekt-Statusrapport finden Sie im ILIAS.

Viel Erfolg sowie spannende und wertvolle Projekterfahrungen wünschen Ihnen

Ihr Dozierendenteam

Begriffe, Abkürzungen

Die verwendeten Klassen und Methoden sollen in englischer Sprache benannt sein. In der folgenden Tabelle sind einige Übersetzungen angegeben.

|  |  |
| --- | --- |
| Engl. Begriff | Dt. Begriff |
| Connect Four  Controller | Vier gewinnt  Spielsteuerung |
| Grid | Spielfeld |
| Opponent  View | Gegenspieler  Sicht |

## Anforderungen

Aus der Aufgabenstellung lassen sich zwei Szenarien ableiten:

Szenario 1:

Als Spieler möchte ich über das Netzwerk gegen einen Gegenspieler „Vier gewinnt“ spielen um Spass zu haben.

Szenario 2:

Als Spieler möchte ich gegen einen Computergegner „Vier gewinnt“ spielen um Spass zu haben.

Systemübersicht

Aus diesen User Stories lässt sich das System visualisieren.

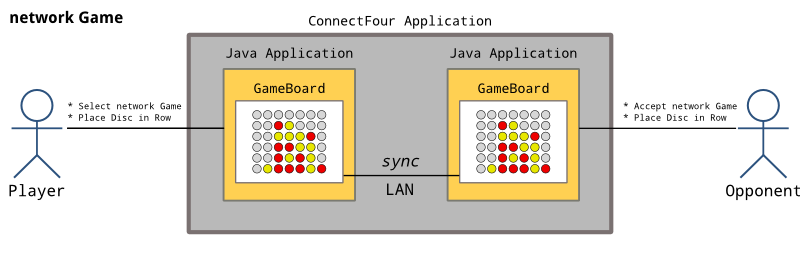


Abbildung 1: Kontextdiagramm des Spiels "Vier Gewinnt" bei einem Spiel gegen einen Gegenspieler über das Netzwerk

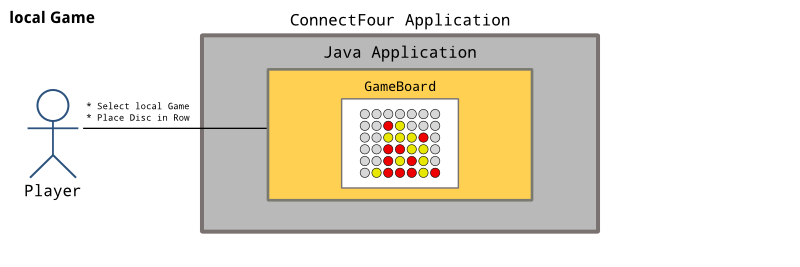


Abbildung 2: Kontextdiagramm des Spiels "„Vier gewinnt“" bei einem Spiel gegen einen Computergegner

Bei der Netzwerkanwendung wird deutlich, dass es ein Spielfelder (Grid) gibt, das jeweils synchronisiert werden muss. Diese Synchronisation ist die einzige Netzwerkaktivität während des Spiels.

Anwendungsfall

Hier ist der Anwendungsfall, welcher aus Szenario 1 abgeleitet wurde.

|  |  |
| --- | --- |
| **Name** | Spiel gegen Netzwerkgegenspieler |
| **Beschreibung** | Netzwerkgegenspieler werden gesucht und danach wird gegen einen dieser Gegenspieler „Vier gewinnt“ gespielt. |
| **Beteiligte Akteure** | User (Player A)  Netzwerkgegenspieler (Player B) |
| **Auslöser** | Zwei Spieler wollen gegeneinander „Vier gewinnt“ spielen. |
| **Vorbedingungen** | Beide Spieler haben die Applikation gestartet  Die beiden Spieler befinden sich im selben Netzwerk |
| **Ergebnis** | Einer der beiden Spieler gewinnt das Spiel oder das Spiel endet unentschieden. |
| **Standardablauf** | 1. Mögliche Gegenspieler werden im Netzwerk gesucht 2. Ein möglicher Gegenspieler wird selektiert, damit „Vier gewinnt“ gespielt werden kann. 3. Wenn der Gegenspieler akzeptiert, wird ein neues Spiel erzeugt (siehe Abbildung 1). 4. Sobald alle Schiffe gesetzt sind, kann der Spieler sein eigenes Spielfeld als bereit markieren. Ab diesem Zeitpunkt können die Schiffe nicht mehr bewegt werden. 5. Sobald beide Spieler bereit sind beginnt das eigentliche Spiel. Player B beginnt. 6. Der Spieler, der an der Reihe ist, setzt seinen Stein in eine Spalte.    * Der Stein fällt herunter    * Sind nun vier gleiche Steine in einer horizontalen, vertikalen oder einer diagonalen Reihe, ist das Spiel beendet und der aktuelle Spieler hat gewonnen.    * Sind alle Spalten besetzt, endet das Spiel unentschieden. 7. Der nächste Spieler ist am Zug und fährt gemäss Punkt 6 fort. |
| **Exceptions**  **(Spielabruch)** | Beide Spieler selektieren sich gleichzeitig für ein Spiel.  Die Verbindung zwischen zwei Spielern wird beendet.  Einer der Spieler beendet die Applikation. |

## Tipps zur Vorgehensweise

Vorgehen:

* Identifizieren Sie die Objekte im Spiel
* Identifizieren Sie die Eigenschaften der Objekte – erstellen Sie daraus die Klassen
* Suchen Sie Beziehungen zwischen den Klassen

Beachten Sie dazu folgende Hinweise

* Entwerfen Sie ihren Design in objektorientierter Denkweise
* Erstellen Sie kleine Klassen, dafür mehrere Klassen
* Beachten Sie Kopplung, Kohäsion und Information Hiding
* Setzen Sie Interfaces ein
* Benutzen Sie die Model-View-Controller-Struktur (siehe Abschnitt 3.1.4)
* Denken Sie ans Testen und Debuggen

## Environment

Für das Spiel über Netzwerk muss Traffic über UDP erlaubt sein, insbesondere müssen UDP-Broadcasts erlaubt sein.

Weil UDP-Broadcast im HSLU-Netz nicht erlaubt sind, wird ein Wifi Access-Point im F‑Stock/Bunker zur Verfügung gestellt. Die Angaben dazu lauten:

* SSID: PRG2\_WLAN
* Passwort: JavaIsFun

Java muss mit (mindestens) der Version 1.7 installiert sein.

## Systemspezifikation

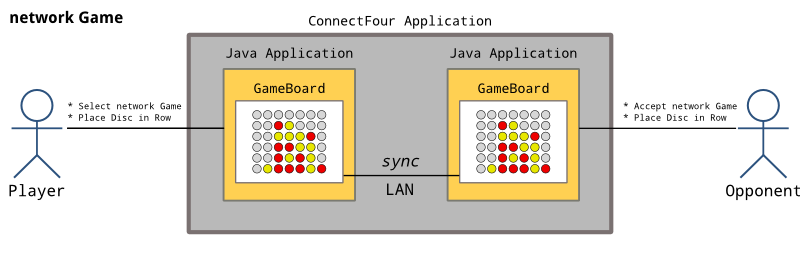
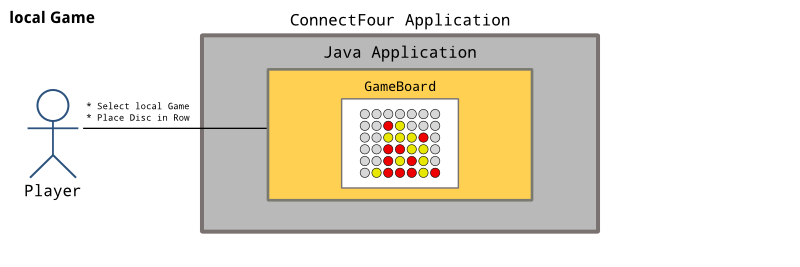
Die folgende Kapiteleinteilung lehnt sich an die "Vier Arten von Sichten” aus [Gernot Starke: "Effektive Softwarearchitekturen: Ein praktischer Leitfaden" (16.01.2014), Seite 80, Kapitel 4.4.2, Bild 4.3[[1]](#footnote-1)](http://books.google.ch/books?id=CaqQAgAAQBAJ&pg=PA80) an.

## Bausteinsichten

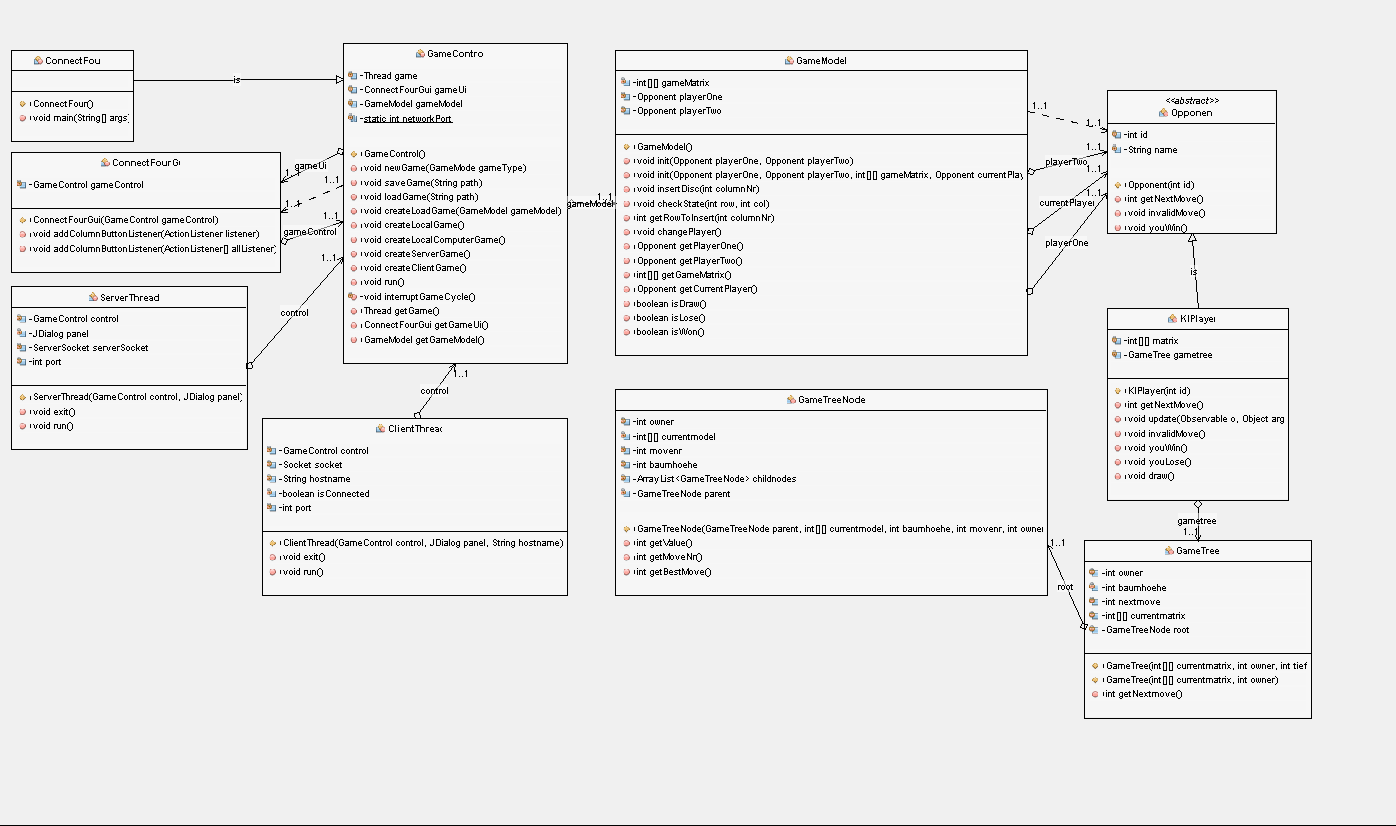
### Systemübersicht

(Studenten) Inhalt dieses Kapitels:

Erstellen Sie ein Diagramm oder Zeichnung das eine Systemübersicht darstellt. Mit Hilfe der Systemübersicht können Sie beispielsweise dem Kunden die Applikation erklären. Die Systemübersicht soll zwei Modi zeigen:

* ein Netzwerk-Spiel gegen einen menschlichen Gegner
* ein lokales Spiel gegen den Computer
* 

### Klassendiagramme.



### MVC

Für die Anwendung wird das Model-View-Controller Entwurfsmuster (MVC) verwendet (siehe (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996) auf Seite 135-143).

#### GameModel

GameModel enthält den aktuellen Zustand des Spiels. Also zum Beispiel die aktuellen Spieler. Das GameModel benachrichtigt die Beobachter, falls es sich geändert hat (z.B. mit dem Observer-Pattern <http://en.wikipedia.org/wiki/Observer_pattern>). Beobachter sind zum Beispiel GameView oder Opponent.

#### GameView

GameView wird durch das GameModel benachrichtigt, falls sich etwas am GameModel geändert hat und stellt anschliessend das GameModel grafisch dar. Also zum Beispiel die aktuellen Spieler, ihren Spielstand auf dem Spielbrett usw. Ausserdem leitet die GameView Eingaben des Benutzers an das GameControl weiter.

#### GameControl

Das GameControl empfängt Kommandos von Opponent oder GameView, entscheidet ob diese gültig sind und darf als einziges das GameModel ändern.

#### Opponent

Opponent wird vom GameModel benachrichtigt, wenn das GameModel geändert hat und darf Befehle an das GameControl senden. Der Opponent besitzt damit viel Ähnlichkeit mit der GameView. Entsprechend können hier gemeinsame Schnittstellen definiert werden.

## Laufzeitsichten

### Zustandsautomat

In rundenbasierten Spielen lässt sich häufig ein Zustandsautomat einsetzen (siehe PRG1/ALG6). Dieser kann durch zwei verschachtelte Zustandsautomaten realisiert werden. Der Äussere kümmert sich um Vorbereitung und Nachbearbeitung des Spiels, während sich der Innere um den Ablauf während des Spiels kümmert.

#### PreparingGame

Das Spielfeld wird aufgebaut und im Menü kann ausgewählt werden zwischen lokalen Spiel gegen einen anderen menschlichen Spieler, lokales Spiel gegen Computer oder Netzwerkspiel gegen einen anderen menschlichen Gegner. Ausserdem kann ein altes Spiel wieder geladen werden.

#### GameRunning

Ein neuer Thread wird erstellt und und das Programm wartet einen Zug des ersten Spielers ab.

#### MyTurn

Der Zug wird, falls validiert ausgeführt (falls invalid MessageBox „ungültiger Zug“ und warten auf neuer Zug) und das Spielfeld upgedatet. Das Spiel auf einen Gewinner und „voll“ überprüft und zum Zustand OpponentTurn oder GameOver übergangen. übergangen.

#### OpponentTurn

Bei menschlichen Spieler faktisch identisch mit MyTurn, ansonsten wird der Zug vom Computer nach Analyse des Spielfeld ausgeführt, oder auf den Zug des Netzwerkspieler gewartet. Der Zug des Netzwerkspielers wird ausgeführt falls valid, falls invalid erscheint die Nachricht ungültiger Zug und muss ein neuer Zug eingegeben werden.

#### WaitingForOpponentResponse

Der Thread wird angehalten bis ein Spalten-Button gedrückt wird oder das AI einen Zug berechnet hat.

#### WaitingForInvitationResponse

Die Netzwerk Funktion wurde vereinfacht. Der Server wartet auf einen beliebigen Client und verbindet sich. Der Server-Spieler führ den ersten Zug aus und sendet ihn an den Client. Der Client führt nächsten Zug aus und sendet ihn an den Server. Daten-Kohärenz nicht gewährleistet.

#### GameOver

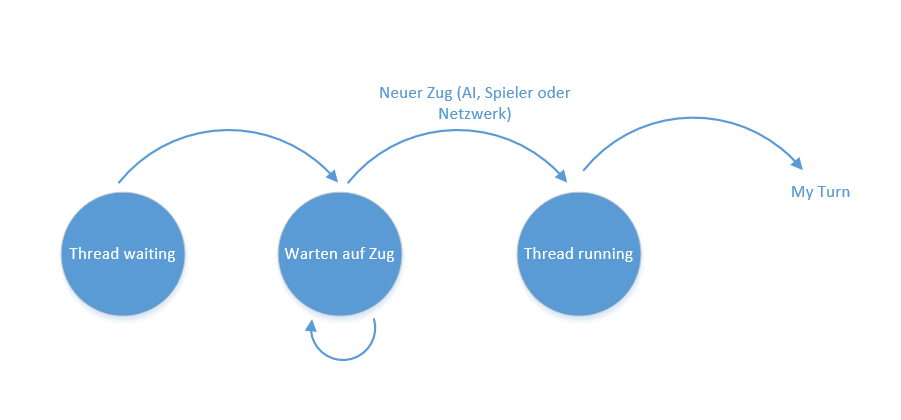
Falls ein Spieler gewonnen hat, wir dies auf dem GUI indiziert. „Glückwunsch Spieler %ID% hat gewonnen“.

### Sequenzdiagramme

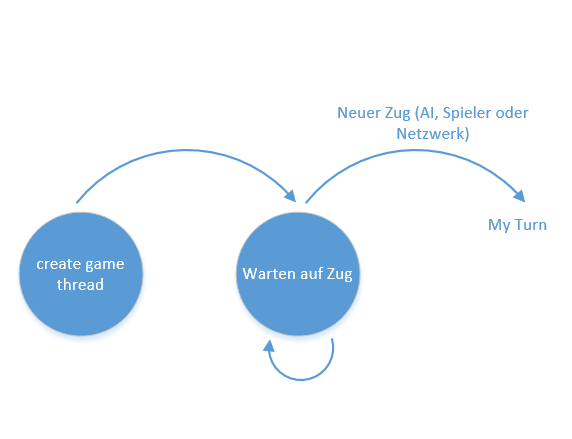
Die Abläufe in den Zuständen können durch Sequenzdiagrammen dargestellt werden (siehe (Buschmann, Meunier, Rohnert, Sommerlad, & Stal, 1996) auf Seite 130-131). Diese zeigen die Interaktionen zwischen den Komponenten.

#### PreparingGame

#### WaitingForOpponentResponse



#### GameRunning



## Verteilungssicht

Das Programm läuft und wurde getestet auf Windows 7 Rechnern.

## Datensicht

Im Model sind folgende Daten gespeichert:

* Spielfeld (Int Array)
* SpielerID (Int)

## Netzwerkprotokoll

Die Netzwerkkommunikation läuft über UDP, der Server sendet den Zug (Spaltennummer als Integer) an den Client. Keine Robustness, es wird gehofft das der Zug ankommt.

# Literaturverzeichnis

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996).   
*Pattern-Oriented Software Architecture Volume 1: A System of Patterns.* Wiley.

1. [www.amazon.de/Effektive-Softwarearchitekturen-Ein-praktischer-Leitfaden/dp/3446436146/](http://www.amazon.de/Effektive-Softwarearchitekturen-Ein-praktischer-Leitfaden/dp/3446436146/) [↑](#footnote-ref-1)