Programmieren & Datenstrukturen

Dokumentation Projekt „Dots & Boxes“

R. Diehl, R. Gisler, M. Klaper, R. Meier

Versionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Datum | Autor | Bemerkungen |
| 1.0.0 | 27.03.2015 | R. Diehl | Initiale Version |
| 1.1.0 | 27.04.2015 | Doz. Team | Überarbeitung aller Kapitel |
| 1.1.1 | 04.05.2015 | Doz. Team | Freigabe |
| 1.2 | 22.05.2015 | LMalushi | Klassendiagramm erstellt |

**Inhalt**

1. Einleitung 2

2. Anforderungen 3

2.1. Regeln „Dots & Boxes“ 3

2.2. Tipps zur Vorgehensweise 4

2.3. Environment 4

3. Systemspezifikation 5

3.1. Bausteinsichten 5

3.1.1. Systemübersicht 5

3.1.2. Komponentendiagramm 5

3.1.3. Klassendiagramme 6

3.1.4. MVC 6

3.2. Laufzeitsichten 7

3.2.1. Zustandsautomat 7

3.3. Verteilungssicht 8

3.4. Datensicht 8

3.5. Netzwerkprotokoll 8

4. Erweiterungsmöglichkeiten 8

# Einleitung

Geschätzte Studierende

Im letzten Teil des Semesters bearbeiten Sie in einem kleinen Team ein Softwareprojekt. Mit diesem Projekt sind die folgenden 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 Sequenzdiagramme lesen und interpretieren.
* Sie können Zustandsdiagramme lesen und interpretieren.
* Sie können Source-Code mit Hilfe von Klassendiagrammen 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 Assistierende 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 [2] finden Sie im ILIAS.

Viel Erfolg sowie spannende und wertvolle Projekterfahrungen wünschen Ihnen

Ihr Dozierendenteam

[1] Projektauftrag Dots & Boxes (dots\_and\_boxes.pdf)

[2] Projekt-Statusrapport (PRG2\_Projekt-Statusrapport.docx)

# Anforderungen

|  |  |
| --- | --- |
| **Name** | Spiel gegen Computer(AI) |
| **Beschreibung** | Ein lokaler Spieler kann gegen den Computer (AI) spielen. |

|  |  |
| --- | --- |
| **Name** | Speichern und laden des Spielstandes gegen Computer |
| **Beschreibung** | Spielstand muss als Datei speicherbar sein. Nach Neustart des Programms, muss der Spielstand wieder verfügbar sein. |

|  |  |
| --- | --- |
| **Name** | Spiel über das Netzwerk |
| **Beschreibung** | Über das Netzwerk kann man gegen einen menschlichen Gegenspieler antreten. Dazu wird die IP Adresse benötigt. |

|  |  |
| --- | --- |
| **Name** | Spielregeln |
| **Beschreibung** | Die Regeln für das Spiel Dots and Boxes müssen gemäss der Dokumentation von Ilse Fischer implementiert werden. |

|  |  |
| --- | --- |
| **Name** | Abgabetermin |
| **Beschreibung** | 28. Mai 2015 bis 12:00 Uhr |

|  |  |
| --- | --- |
| **Name** | Benutzeroberfläsche |
| **Beschreibung** | Es wird ein GUI erstellt. In diesem sind die aktuellen Spieler, der aktuelle Spielstand sowie die Information für den Spieler, welcher den nächsten Zug setzen kann. |

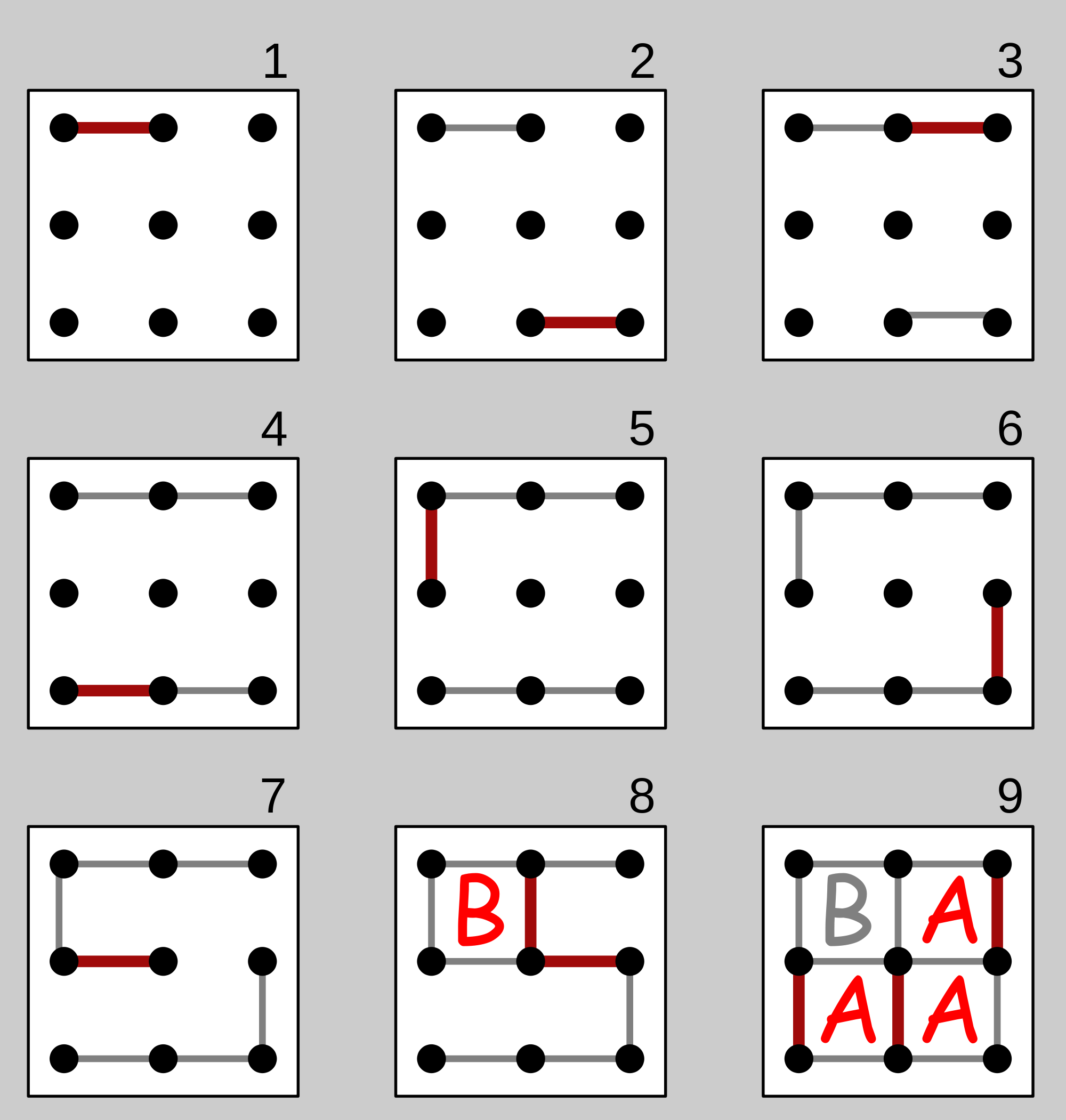
## Regeln „Dots & Boxes“

Zwei Spieler beginnen mit einer quadratischen (oder rechteckigen) Anordnung von Punkten.

Ein Zug besteht darin, zwei horizontal oder vertikal benachbarte Punkte durch einen Strich zu verbinden.

Hat ein Spieler, eine Spielerin bei ihrem Zug ein Einheitsquadrat vollendet, so kennzeichnet er/sie dieses Kästchen (z.B. füllt das Kästchen mit der Strichfarbe des Spielers) und macht einen weiteren Strich.

Das Spiel ist zu Ende, wenn alle Kästchen vollendet sind. Die Siegerin, der Sieger ist derjenige oder diejenige mit den meisten Kästchen.



## 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 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 Testing und Debugging

## 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

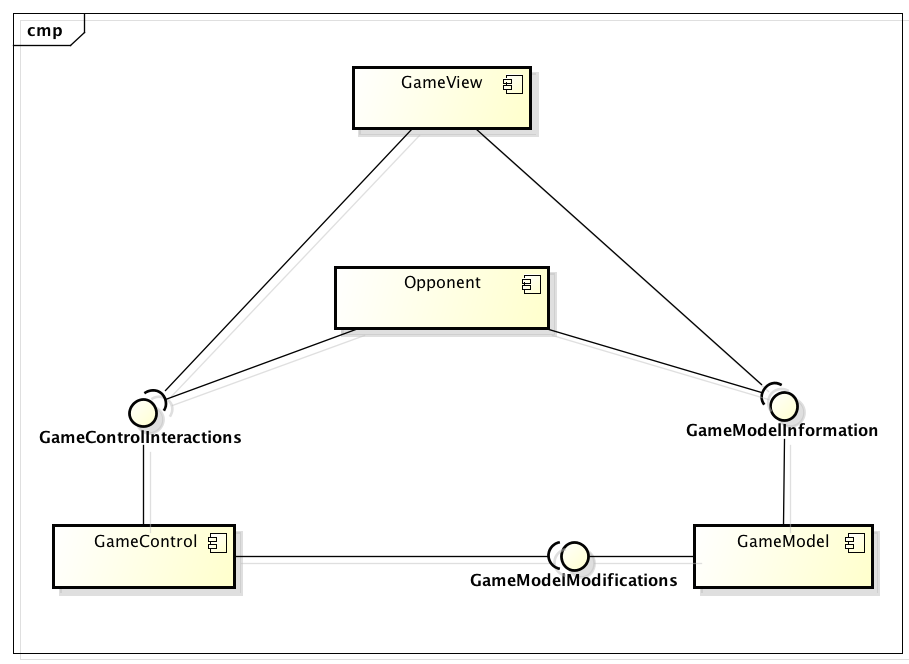
Die folgende Kapiteleinteilung lehnt sich an die “Vier Arten von Sichten” aus [Gernot Starke: "Effektive Softwarearchitekturen: Ein praktischer Leitfaden" (16.01.2014), Seite 80, Kapitel 4.4.2, Bild 4.3[[1]](#footnote-1)](http://books.google.ch/books?id=CaqQAgAAQBAJ&pg=PA80) an.

## Bausteinsichten

### Systemübersicht



### Komponentendiagramm

Dieses Diagramm zeigt einen möglichen Aufbau der Applikation. Opponent kann ein lokaler Computer-Opponent oder ein entfernter Netzwerk-Opponent sein. Bitte beachten Sie, dass die Namensgebung sehr allgemein gehalten ist und auf Ihr konkretes Spiel angepasst werden soll.

* + - * 1. GameControlInteractions

Über diese Schnittstellen können Opponent und GameView Befehle an die GameControl absetzen (z.B. Neues Spiel, Gegner ausgewählt, etc.).

* + - * 1. GameModelInformation

Interessierte Klassen werden über diese Schnittstelle über Änderungen am GameModel informiert.

* + - * 1. GameModelModifications

Über diese Schnittstelle kann die GameControl Änderungen am GameModel vornehmen.

### Klassendiagramme

Für bessere Einsicht bitte zoomen.



### Gameboard

GUI des Rasters: Stellt mit Hilfe der BoxPicture-, LinePicture- und DotPicture-Klassen das eigentliche Spielfeld dar. Für die Anordnung wird ein GridBagLayout verwendet.

#### BoxPicture

GUI der Box-Klasse: Jede Box wird als Attribut an ein BoxPicture übergeben. Diese Klasse übernimmt dann die visuelle Darstellung der Box.

#### LinePicture

GUI der Line-Klasse: Jede Line wird als Attribut an ein LinePicture übergeben. Diese Klasse übernimmt dann die visuelle Darstellung der Line. Dieses Objekt reagiert auf MouseClicked, MouseEntered und MouseExited (für das Hervorheben der Linie, wenn die Maus darüber fährt).

#### DotPicture

GUI für einen Punkt. Wird nur für die Darstellung des Rasters verwendet.

### MVC

Für die Anwendung wird das Model-View-Controller Entwurfsmuster (MVC) verwendet.

(Siehe: <http://de.wikipedia.org/wiki/Model_View_Controller>)

#### 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, ihr Spielstand auf dem Spielbrett usw. Ausserdem leitet die GameView Eingaben des Benutzers an das GameControl weiter.

#### GameControl

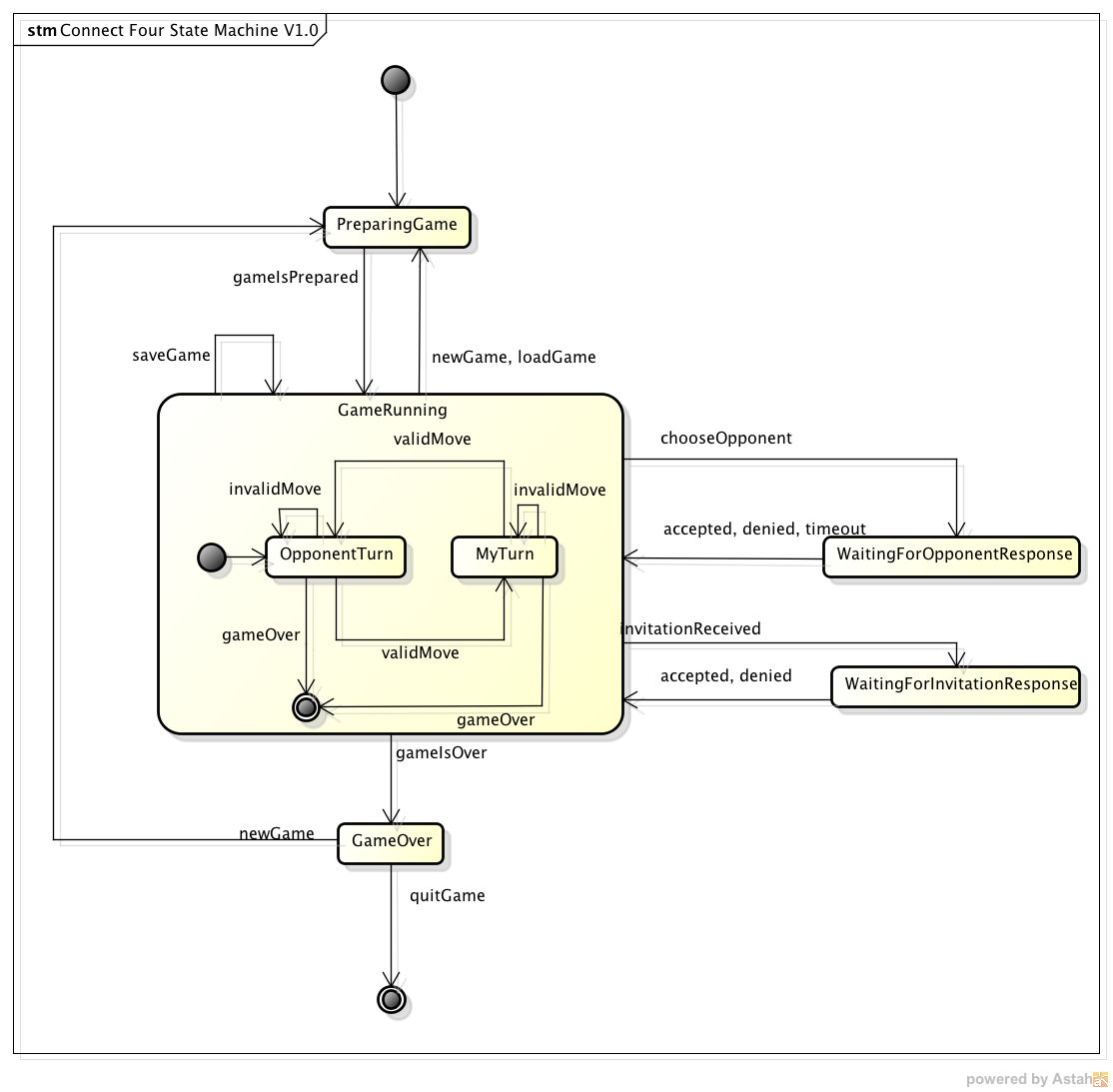
Das GameControl empfängt Kommandos von Opponent oder GameView, entscheidet ob diese gültig sind und darf als einziges das GameModel ändern.

#### 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. Dies 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 dem Spiel kümmert.

#### PreparingGame

Das Spielfeld wird aufgebaut und die Teilnehmenden werden initialisiert. Der/die erste Spieler/in ist ein Mensch, der/die zweite wird zunächst auf einen Computergegner gesetzt, damit sofort mit dem Spiel begonnen werden kann. Nach dem Aufbau des Spielfelds geht der Zustand per gameIsPrepared unmittelbar in den Zustand GameRunning (bzw. konkret direkt in OpponentTurn) über.

#### GameRunning

Teilnehmer sind initialisiert. Die Domain kontrolliert welcher Spieler an der Reihe ist. Der Spieler der am Zug ist kann diesen tätigen. Nach dem Ausführen des Zuges wird dieser validiert. Spieler werden informiert, wer als nächstes an der Reihe ist.

#### MyTurn

Die Spieler sehen das graphische Spielfeld. Der aktuelle Spieler wählt eine Linie aus. Das GUI übergibt die gewählten Infos an die Domain. Die Domain wertet diese anschliessend aus.

#### OpponentTurn

Der Gegenspieler oder die AI wählt eine Linie aus. Diese Line wird an der Domian übergeben. Die Domaine validiert den Entscheid auf ihre Richtigkeit.

#### WaitingForOpponentResponse

Es werden alle Verbindungsanfragen empfangen. Wartet auf Gegenspieleranfragen. Sendet diese ans GUI weiter.

#### WaitingForInvitationResponse

Falls über das Netzwerk eine Anfrage zur Teilnahme an einem anderen Spiel eintrifft, geht der Zustand von “GameRunning” (bzw. konkret einem seiner Unter-Zustände OpponentTurn oder MyTurn) per invitationReceived über in WaitingForInvitationResponse. Dort wird ausgewählt, ob die Einladung angenommen werden soll oder nicht. Unabhängig von dieser Entscheidung geht der Zustand danach per accepted, denied wieder in GameRunning über.

#### GameOver

Spieler löst im GUI eine GameOver Message aus. Alle bestehenden Verbindungen beendet.

## Verteilungssicht

Die Applikation läuft entweder auf einem Rechner oder auf zwei wenn es ein Netzwerkspiel ist.

## Datensicht

 (Studenten) Inhalt dieses Kapitels:

* Wie sieht die Applikation aus Datensicht aus?
* Was für Daten werden vor der Applikation verwaltet, geteilt, gespeichert?

## Netzwerkprotokoll

Der Server baut eine TCP-Verbindung mit dem Client auf und übergibt den Spielernamen. Falls der Gegenspieler das Spiel akzeptiert hat, sendet er seinen Spielernamen zurück. Danach werden die Spielzüge und war das daran ist über das Netzwerk übertragen.

# Erweiterungsmöglichkeiten

 (Studenten)

* Eigene Erweiterungen dokumentieren

Folgende Erweiterungen sind für das Projekt denkbar:

* Variable Grösse des Spielfelds
* Computergegner gegeneinander antreten lassen
* Der Verlauf einer Partie kann aufgezeichnet und wieder abgespielt werden

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)