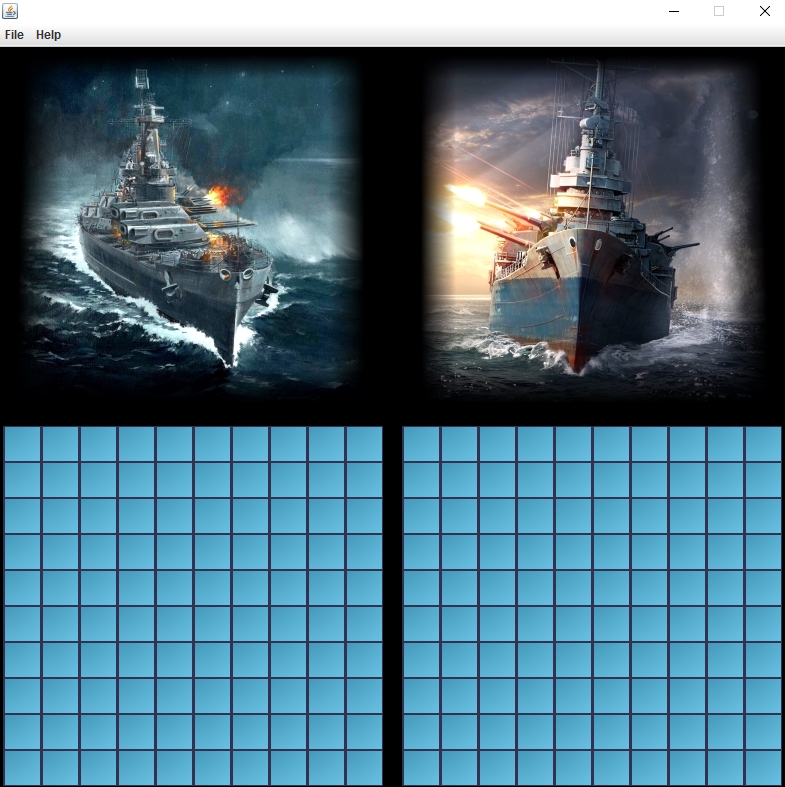
Schiffeversenken

Programmieren & Datenstrukturen

Dokumentation Projekt "Schiffe versenken"

R. Diehl, M. Klaper, R. Meier, P. Sollberger, N. Schär, T. vor der Brück

Versionen:

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Datum | Autor | Bemerkungen |
| 1.0 | 21.03.2016 | Dozierendenteam | Initiale Version |

**Inhalt**

1. Einleitung 3

1.1. Begriffe, Abkürzungen 3

2. Anforderungen 4

2.1. Systemübersicht 4

2.2. Anwendungsfall 5

2.3. Tipps zur Vorgehensweise 7

2.4. Environment 7

3. Systemspezifikation 8

3.1. Bausteinsichten 8

3.1.1. Systemübersicht 8

3.1.2. Klassendiagramme 9

3.1.3. Klassenbeschreibung 10

3.2. Laufzeitsichten 11

3.2.1. Zustandsautomat 11

3.2.2. Sequenzdiagramm 12

3.3. Weitere Pendenzen 12

4. Literaturverzeichnis 12

# 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 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 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 |
| Battleship  Controller | Schiffe versenken  Spielsteuerung |
| PlayingField | Spielfeld |
| Enemy  View | Gegenspieler  Sicht |

## Anforderungen

Aus der Aufgabenstellung lassen sich zwei Szenarien ableiten:

Szenario 1:

Als Spieler möchte ich über das Netzwerk gegen einen Gegenspieler Schiffe versenken spielen um Spass zu haben.

Szenario 2:

Als Spieler möchte ich gegen einen Computergegner Schiffe versenken spielen um Spass zu haben.

Systemübersicht

Aus diesen User Stories lässt sich das System visualisieren.

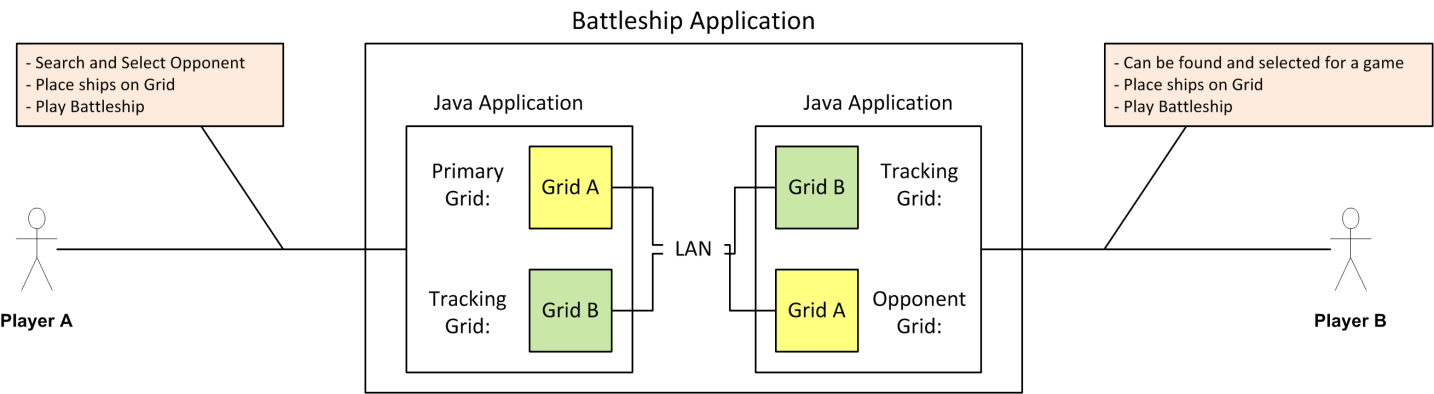


Abbildung 1: Kontextdiagramm des Spiels "Schiffe versenken" (englisch: Battleship) bei einem Spiel gegen einen Gegenspieler über das Netzwerk

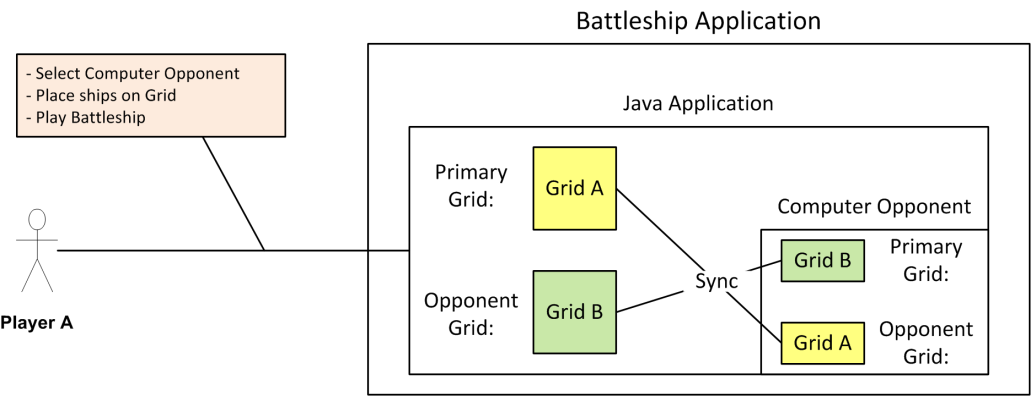


Abbildung 2: Kontextdiagramm des Spiels "Schiffe versenken" bei einem Spiel gegen einen Computergegner

Bei beiden Anwendungen wird deutlich, dass es vier Spielfelder (Grids) gibt, wobei jeweils zwei miteinander synchronisiert werden müssen. 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 "Schiffe Versenken" gespielt. |
| **Beteiligte Akteure** | User (Player A)  Netzwerkgegenspieler (Player B) |
| **Auslöser** | Zwei Spieler wollen gegeneinander "Schiffe Versenken" spielen. |
| **Vorbedingungen** | Beide Spieler haben die Applikation gestartet  Die beiden Spieler befinden sich im selben Netzwerk |
| **Ergebnis** | Einer der beiden Spieler gewinnt das Spiel. |
| **Standardablauf** | 1. Mögliche Gegenspieler werden im Netzwerk gesucht 2. Wenn ein Gegenspieler gefunden wird, wird ein neues Spiel erzeugt (siehe Abbildung 1). 3. Beide Spieler setzen ihre Schiffe auf dem eigenen Spielfeld. Dabei gelten folgende Regeln:  * Jedes Schiff besetzt eine bestimmte Anzahl von horizontal oder vertikal aufeinanderfolgenden Feldern (d.h. nicht diagonal). * Die Schiffe können sich nicht überlappen * Beide Spieler haben dieselben Schiffe   Folgende Schiffe stehen jedem Spieler am Beginn des Spieles zur Verfügung:   * 1 Schlachtschiff (4 Kästchen) ☐☐☐☐☐ * 1 Kreuzer (3 Kästchen) ☐☐☐☐ * 2 Zerstörer (2 Kästchen) ☐☐☐ * 2 U-Boote (1 Kästchen) ☐☐  1. Sobald alle Schiffe gesetzt sind, kann das Spiel gestartet werden. 2. Sobald beide Spieler bereit sind beginnt das eigentliche Spiel. 3. Der Spieler, der an der Reihe ist, attackiert ein Feld des Gegners. Bei einem Angriff gelten folgende Regeln:  * Der Gegner signalisiert ob sich auf diesem Feld ein Schiff befindet (Treffer) oder ob es Meer ist (Fehlschuss). * Bei einem Treffer darf der attackierende Spieler nochmals schiessen. * Bei einem Fehlschuss ist der andere Spieler dran. * Sobald alle Felder eines Schiffs getroffen worden sind, gilt dieses als versenkt. * Der attackierte Spieler teilt bei einem Treffer mit, ob sein Schiff versenkt ist.  1. Punkt 6 wird solange wiederholt, bis bei einem Spieler alle Schiffe versenkt wurden. Dann ist das Spiel vorbei und der Spieler, welcher noch Schiffe besitzt, hat gewonnen. |
| **Exceptions**  **(Spielabruch)** | 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

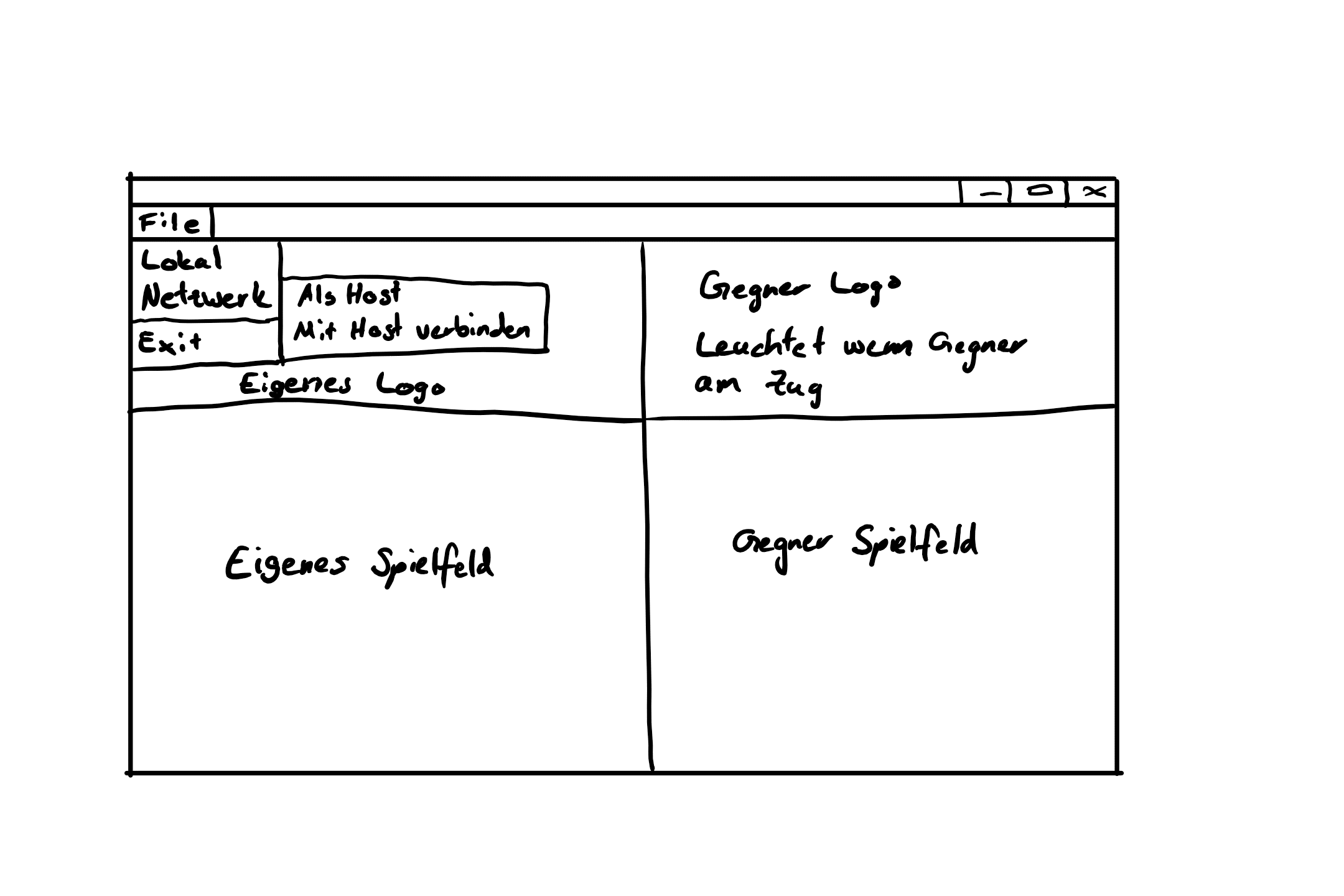
Java muss mit (mindestens) der Version 1.7 installiert sein.

## 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



Dieses Bild zeigt die ungefähre Sielansicht. Über File kann ein neues Lokales oder Netzwerk Spiel gestartet werden. Zudem kann das Fenster über Exit oder auch das Kreuz oben rechts geschlossen werden.

Das Fenster ist in zwei Teile aufgeteilt. Links ist das eigene Spielfeld und rechts das des Gegners. Oben auf beiden Seiten wird ein Logo angezeigt, welches Leuchtet wenn man am Zug ist (Wenn linkes leuchtet ist man selbst am Zug).

Zu beginn des Spiels können auf dem eigenen Feld die eingenen Schiffe gesetzt werden. Danach kann, je nachdem wer am Zug ist, auf dem gegnerischen Spielfeld schüsse abgegeben werden. Dabei gelten folgende Farbregeln:

* Hell Blau -> Wasser getroffen
* Dunkel Blau -> Wasser
* Gelb -> Schiff getroffen
* Rot -> Schiff versenkt, Schiff plazieren (Plazierungs konflikt)
* Scharz -> Eigenes Schiff
* Grün -> Schiff plazieren (vorschau)

### Klassendiagramme



### Klassenbeschreibung

Für die Anwendung wurde die Strukturierung in mehrere Klassen unterteilt. Um dies möglichst mit wenig Kopplung zu tun, wurden Interfaces als Schnittstellte eingefügt.

#### IPlayingField

Das IPlayingField enthält Funktionen, mit der auf das Gui zugegriffen werden kann. So kann z.B. der aktuelle Spieler angezeigt oder auch die Treffer angezeigt werden.

#### ILogic

ILogic bietet Funktionen, mit der das Gui die Schüsse abfragen kann (ob es getroffen wurde, usw). Zudem können auch die Schiffe am Anfang gesetzt werden.

#### ILogicEnemy

Über dieses Interface kann die Netzwerkverbindung ihre Daten, die sie vom Gegner erhalten hat zurückschicken.

#### IEnemy

Dies ist die Schnittstelle, die die Logik anspricht, wenn sie einen Schuss an den Gegner sendet. Dies passiert dann z.B. über eine aktive Netzwerkverbindung

#### INetwork

Dieses Interface bietet die Funktionen um eine Netzwerkverbindung aufbauen zu können.

## 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 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

1. Modus im Menü auswählen (Lokal oder Netzwerk)
2. Startspieler wird per Zufall festgelegt.
3. Schiffe platzieren. Der Spieler kann seine Schiffe in seinem Spielfeld setzen.
4. Wenn alle Schiffe platziert sind, wird entweder in den „MyTurn“ oder „OpponentTurn“ gewechselt.

#### MyTurn

1. Ich kann im Feld des Gegners einen Schuss abgeben und wechsle in den Zustand „WaitingForOpponentResponse“.
2. GUI aktualisieren.

#### OpponentTurn

1. Ich warte auf die Koordinaten des Gegners.
2. Bei Treffer: Bleibe nochmals bei OpponentTurn
3. Bei keinem Treffer: gehe zu MyTurn
4. GUI aktualisieren.

#### WaitingForOpponentResponse

1. Die Koordinaten werden übergeben und ausgwertet.
2. Bei Treffer: Gehe zu MyTurn
3. Bei keinem Treffer: gehe zu OpponentTurn

#### WaitingForInvitationResponse

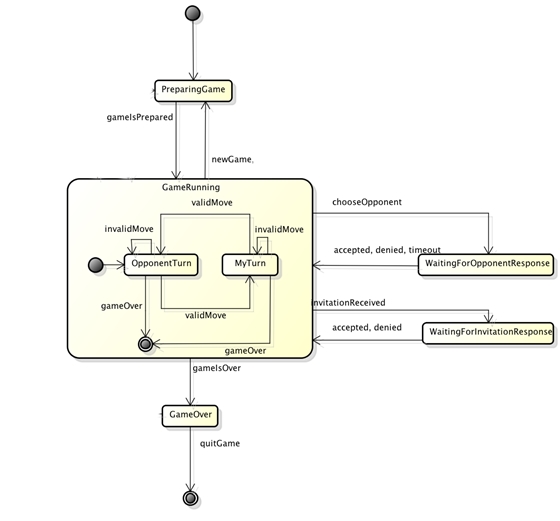
Falls über das Netzwerk eine Anfrage zur Teilnahme an einem anderen Spiel eintrifft, geht der Zustand von "GameRunning” (bzw. konkret einem seiner Unter-Zustände OpponentTurn oder MyTurn) per invitationReceived über in WaitingForInvitationResponse. Dort wird ausgewählt, ob die Einladung angenommen werden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

#### GameOver

1. Wenn alle Schiffe getroffen wurden, wird den Spielern mitgeteilt ob sie gewonnen oder verloren haben.

### Sequenzdiagramm

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.



## Weitere Pendenzen

Leider konnten wir aus Zeitlichen Gründen das Programm nicht vollständig beenden. So kann zwar eine Verbindung (auch mit Broadcast) aufgebaut werden, allerdings muss zuerst ein Host gestartet werden, bevor eine Verbindung gesucht wird. Ansonsten stürzt das Programm ab.

Zudem ist die KI nicht vollständig ausprogrammiert. Das Starten eines Spiels gegen eine KI funktioniert, allerdings werden momentan die Schiffe der KI immer gleich gesetzt und die Schüsse sind immer Zufällig.

## 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)