Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Technik & Architektur

Programmieren & Datenstrukturen

Dokumentation Projekt „Dots & Boxes“

R.Mattmann, D.Heer, K.Bieri, T.Linder

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
1.1.2	21.05.2015	T. Linder	- Anforderungen erweitert - Entwurfsdiagram eingefügt
1.1.3	22.05.2015	T. Linder	- Datensicht, Datenmodell eingefügt

Inhalt

1. 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)

2. Anforderungen

Der Projektauftrag wurde handschriftlich abgegeben.

Es muss ein Spielfeld aus GUI Komponenten aufgebaut werden. Die Events des GUI's müssen an eine Gamelogik weitergegeben werden. Es ist notwendig mit EventListener zu arbeiten. Nach einem ersten Überblick, sind wir zum Schluss gekommen, dass wir auch mit Threads arbeiten müssen. Die Zusammenhänge der werden mit Interfaces umgesetzt.

Zusätzlich zu den ganzen Programmieraufgaben werden wir uns auch noch mit dem Sourcecontroltool Git beschäftigen.

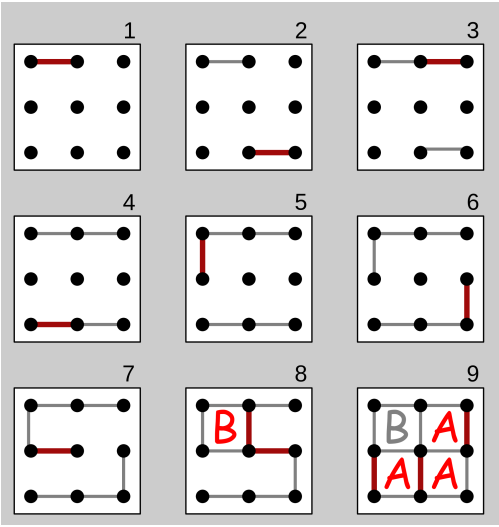
2.1. Regeln „Dots & Boxes"

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.



2.2. 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

2.3. 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.

3. 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] *an.*

3.1. Bausteinsichten

3.1.1. Systemübersicht

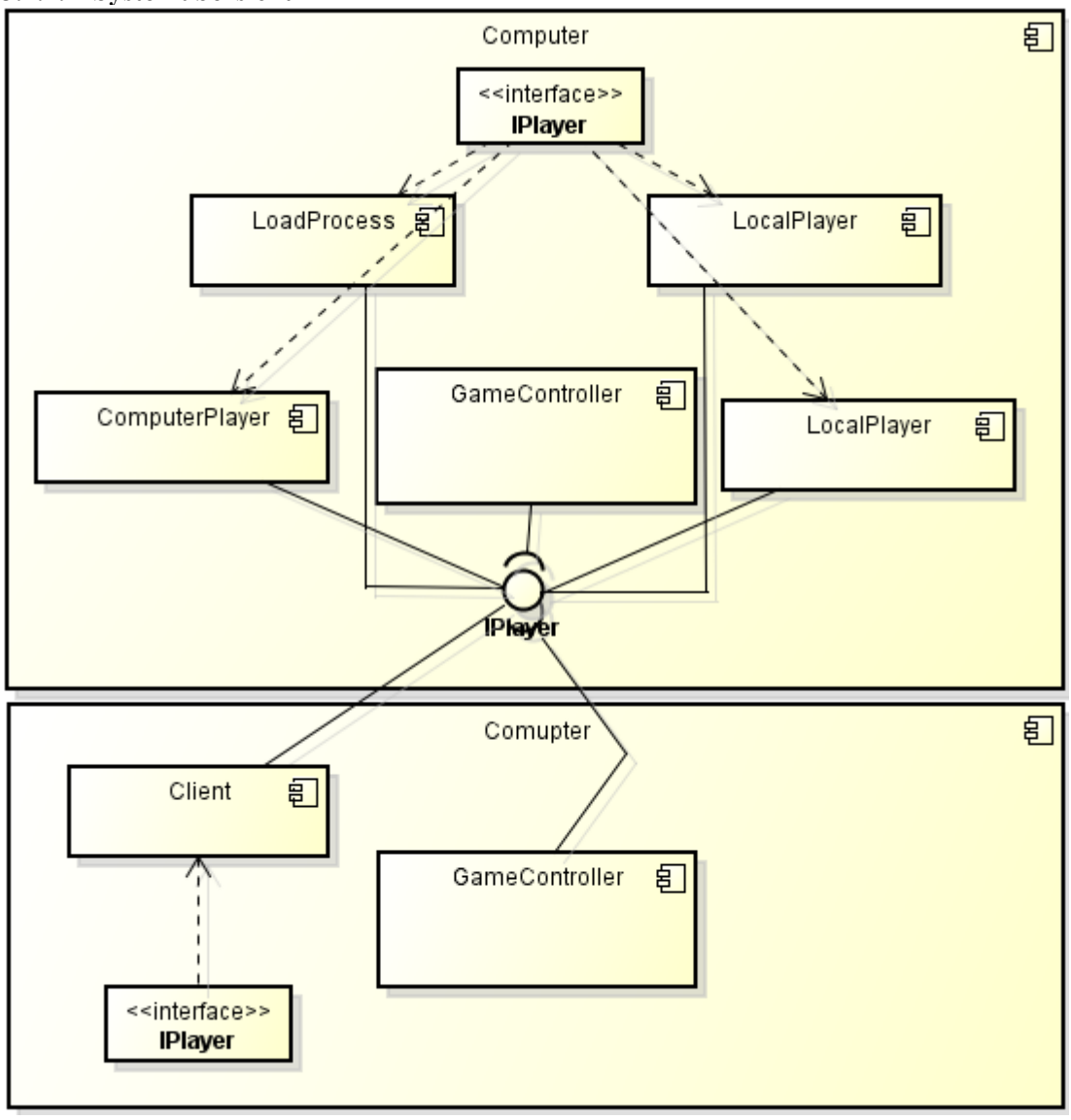


Abbildung 1: Aufbau des Systems

1 www.amazon.de/Effektive-Softwarearchitekturen-Ein-praktischer-Leitfaden/dp/3446436146/

3.1.2. Komponentendiagramm

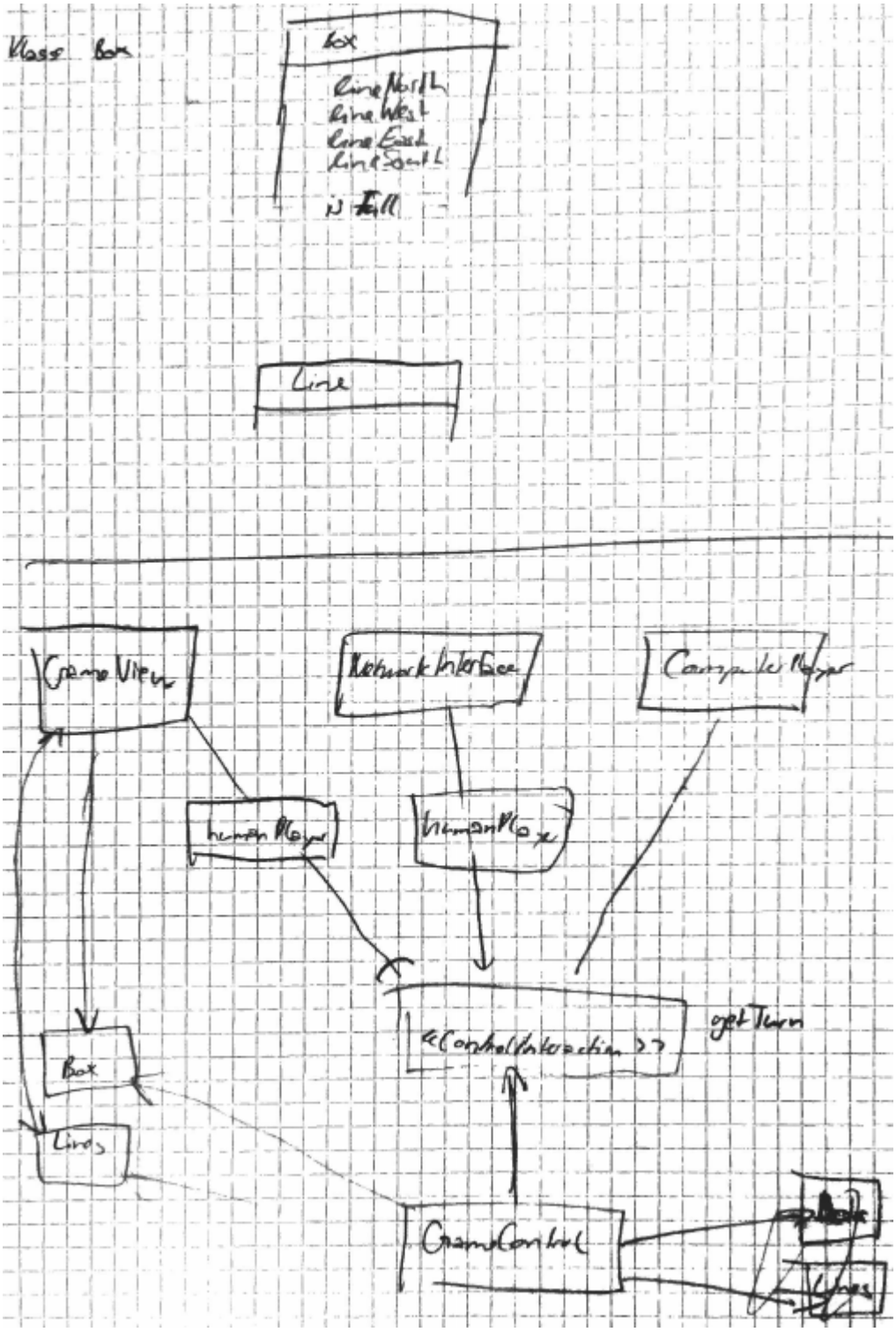


Abbildung 2: Entwurf der Komponenten

In der Abbildung ist der erste Entwurf der Klassen die wir einsetzen aufgezeichnet. Bis zum Schluss konnten wir uns an die Definierte Struktur halten. In den Klassendiagrammen wird die Struktur wieder erkennbar sein.

Interface Player

Mit Hilfe des Interfaces Player (definiert die Methode getTurn) kann jede Klasse die dieses Interface implementiert in die GameController Klasse integriert werden.

GameController

Die GameController ist so aufgebaut, dass sie sich nur mit den Typen des Interface Player beschäftigt. Somit ist eine lose Kopplung zu jeder Klasse die als Spieler teilnehmen will gegeben. Dies kann auch über Netzwerk umgesetzt werden.

Mit diesem Konzept ist es uns möglich auch den Ladevorgang als normales Spiel abzuarbeiten.

3.1.3. Klassendiagramme

In allen Diagrammen wird eine Spielsituation dargestellt. Das Interface IPlayer wie auch die Klasse Player sind in jeder Abbildung vorhanden. Die GameController Klasse arbeitet mit diesem Interface. Jeder potenzielle Spieler muss das Interface implementieren. Ist diese Situation gegeben kann ein Spieler ohne Probleme im Game integriert werden.

3.1.3.1. Allgemeine Spielsituation

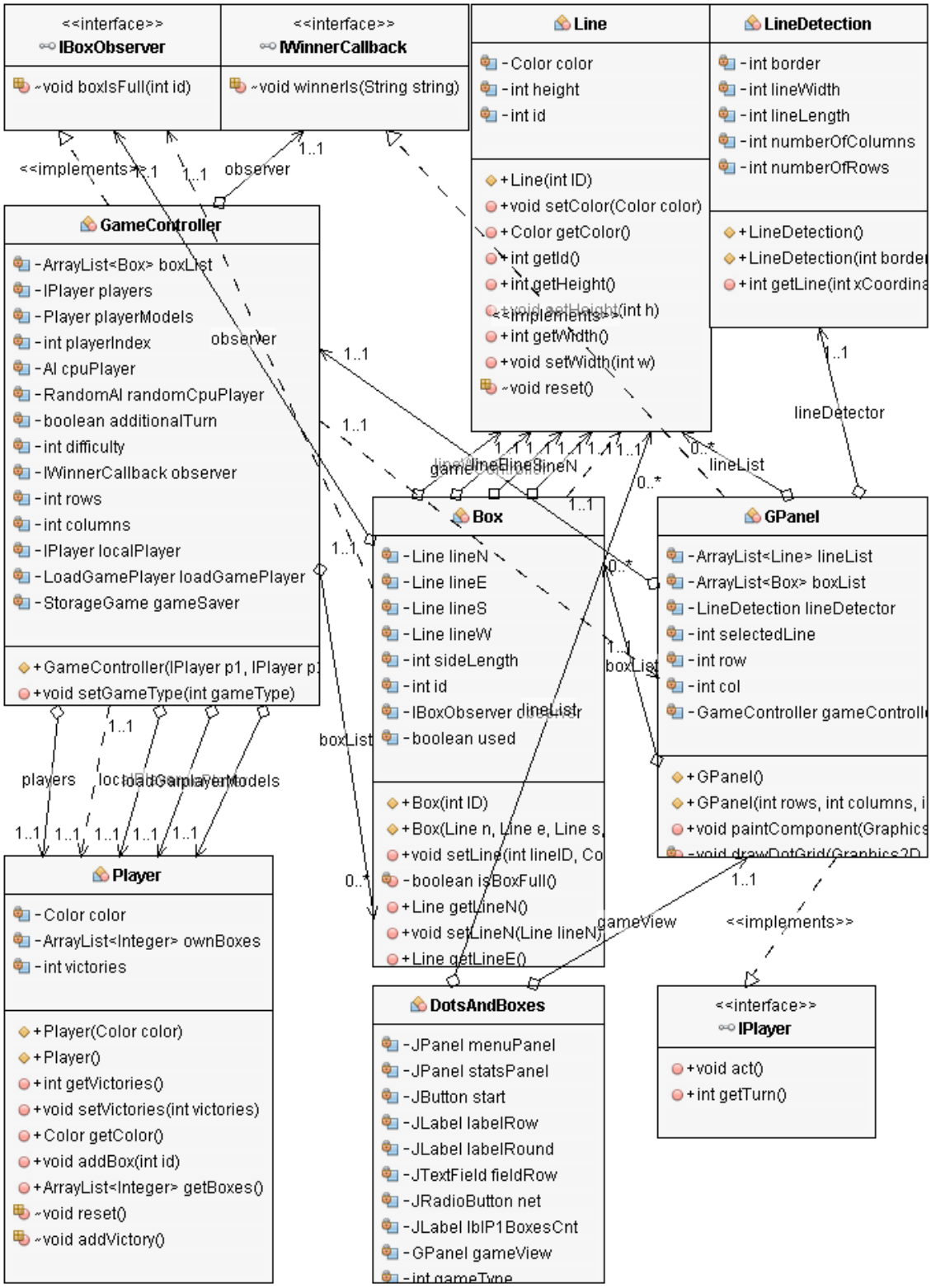


Abbildung 3: Allgemeines Klassendiagramm

3.1.3.2. Künstliche Intelligenz

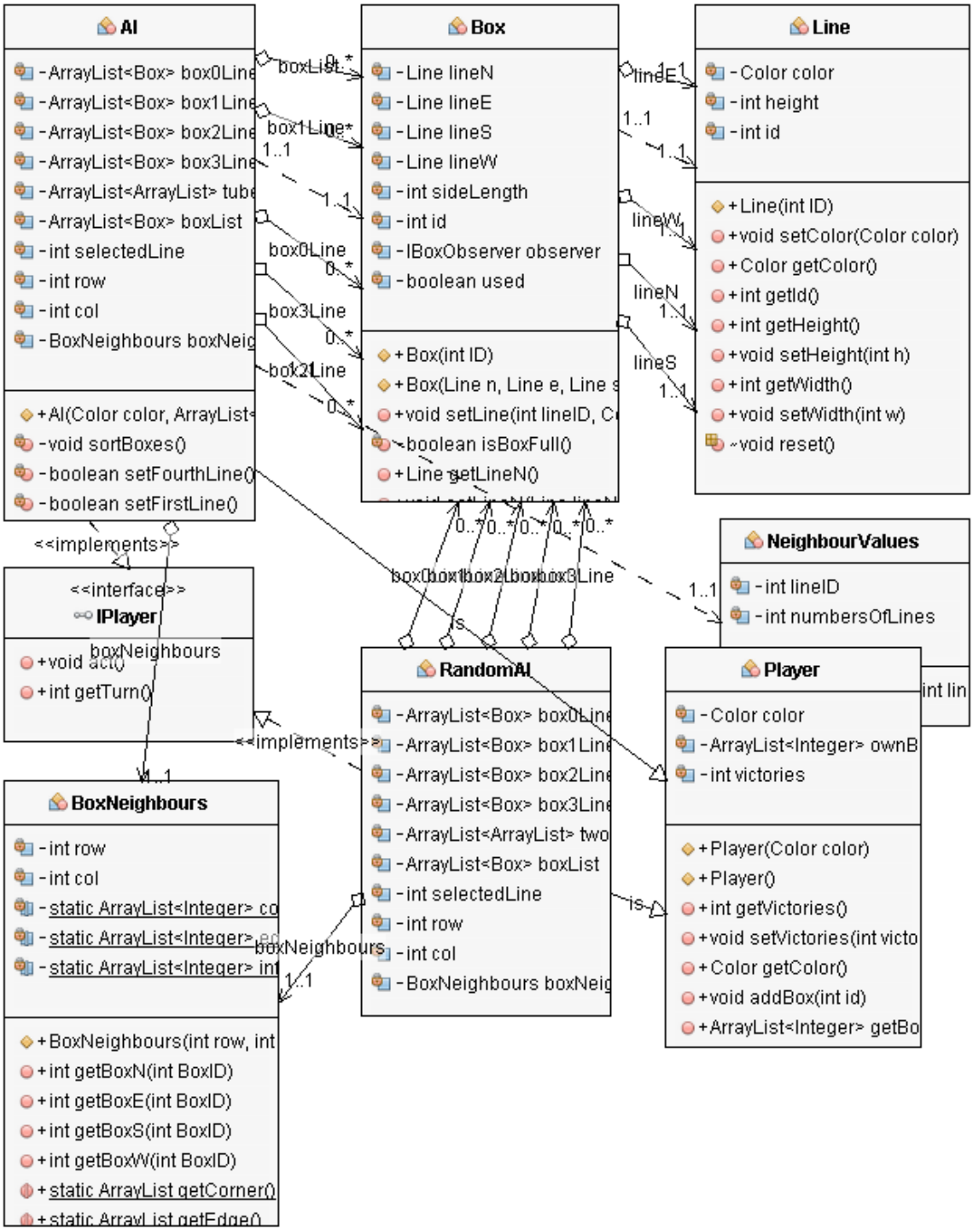


Abbildung 4: KI Klassendiagramm

Spielt man gegen den Computer so hat man zwei Möglichkeiten.

Wählt man „easy“ so wählt die Math.random()-Methode bei jedem Spielzug eine zufällige Box und prüft jene auf unmarkierte Linien. Ist bei dieser Box keine Linie mehr frei wird eine neue Box „ausgewählt“.

Wählt man die schwierigkeitsstufe „not Random“ so beginnt der Computergegner überlegte Spielzüge zu machen. Zuerst sortiert er alle Boxen nach anzahl markierten Linien. Anschliessend prüft er ob eine Box drei markierte Linien hat und schliesst diese.
Falls keine Box drei Linien durchläuft er die ArrayList mit Boxen mit einer Linie. Dort überprüft er alle Nachbarn. Bevorzugt werden Randlinien gefolgt von Nachbarsboxen mit einer oder null markierten Linien. Solche mit zwei werden ignoriert.
Falls keine zweite Linie gesetzt werden konnte werden die noch unmarkierten Boxen geprüft, mit den selben Kriterien wie bei den Boxen welche eine markierte Linie haben.
Wenn es auch keine leeren Boxen mehr hat, sollte es auf dem Spielfeld nur noch solche Boxen mit zwei oder vier markierten Linien haben.
Wenn man die dritte Linie einer Box markiert, kann der Gegenspieler die Box im nächsten Spielzug schliessen und einen Punkt gewinnen. Da man nach dem schliessen noch einer Box nochmals eine Linie setzen darf, könnte man so eine sogenannte Röhre füllen. Eine Röhre sind mehrere aneinander liegende Boxen mit zwei markierten Linien. Die grenzlinie ist jeweils noch nicht markiert. Bei einer bei einer ungeraden anzahl Linien schliesst er die kleinste. Bei einer ungeraden jeweils die Zweitkleinste, so dass er am schluss die grösste Röhre schliessen kann.

3.1.3.3. Netzwerkspiel

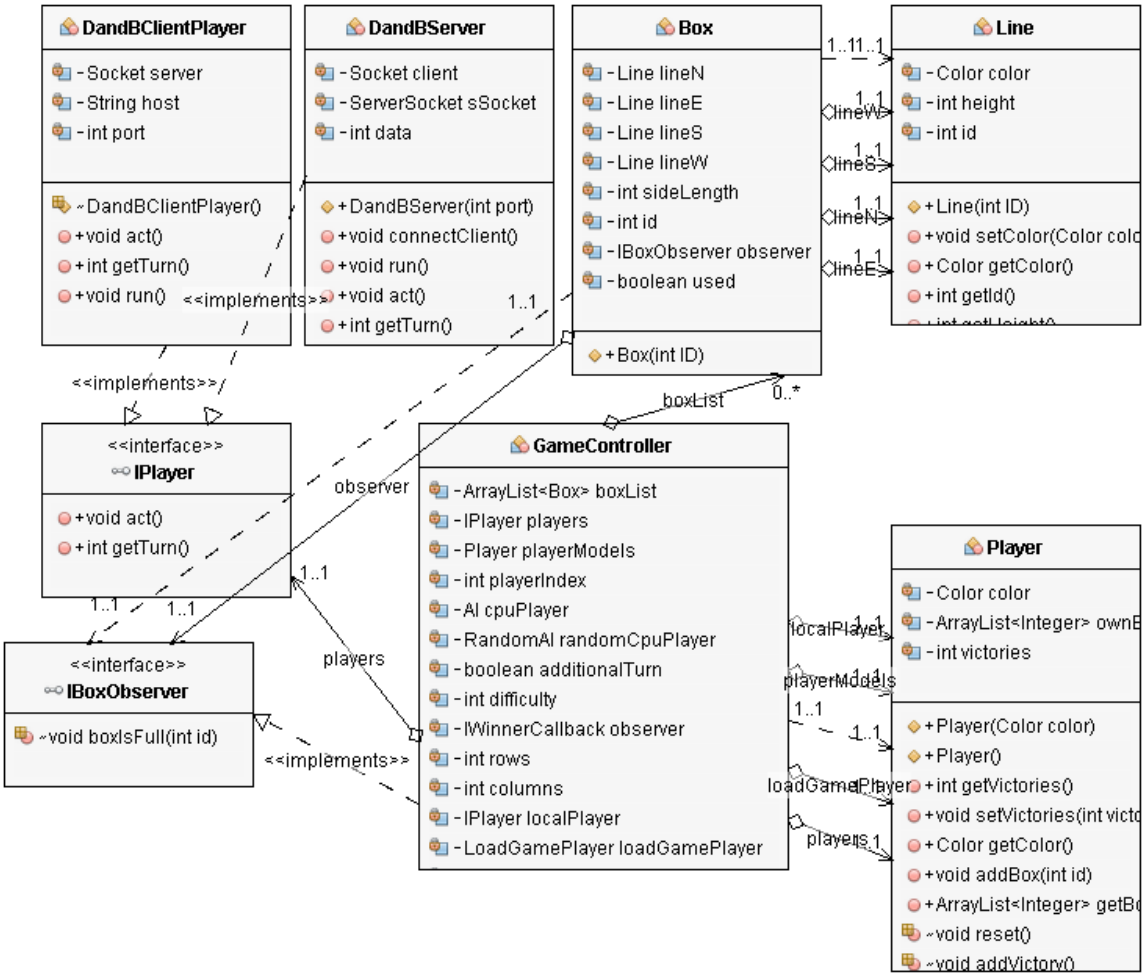


Abbildung 5: Netzwerkspiel Klassendiagramm

3.1.3.4. Load and Save

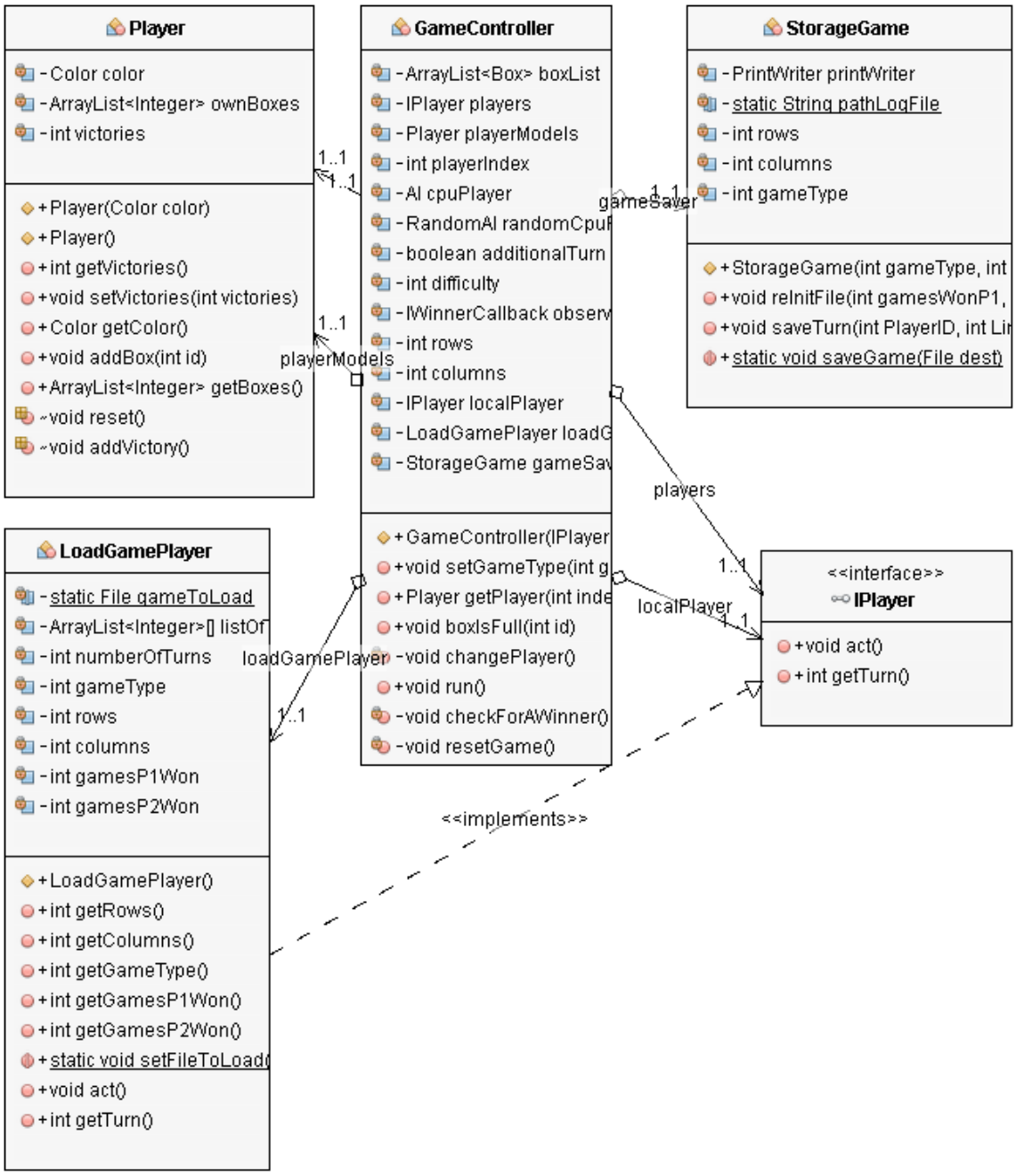


Abbildung 6: Load and Save Klassendiagramm

3.1.4. MVC

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

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

3.1.4.1. 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.

3.1.4.2. 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.

3.1.4.3. GameControl

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

3.1.4.4. 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.

3.2. Laufzeitsichten

3.2.1. Zustandsautomat

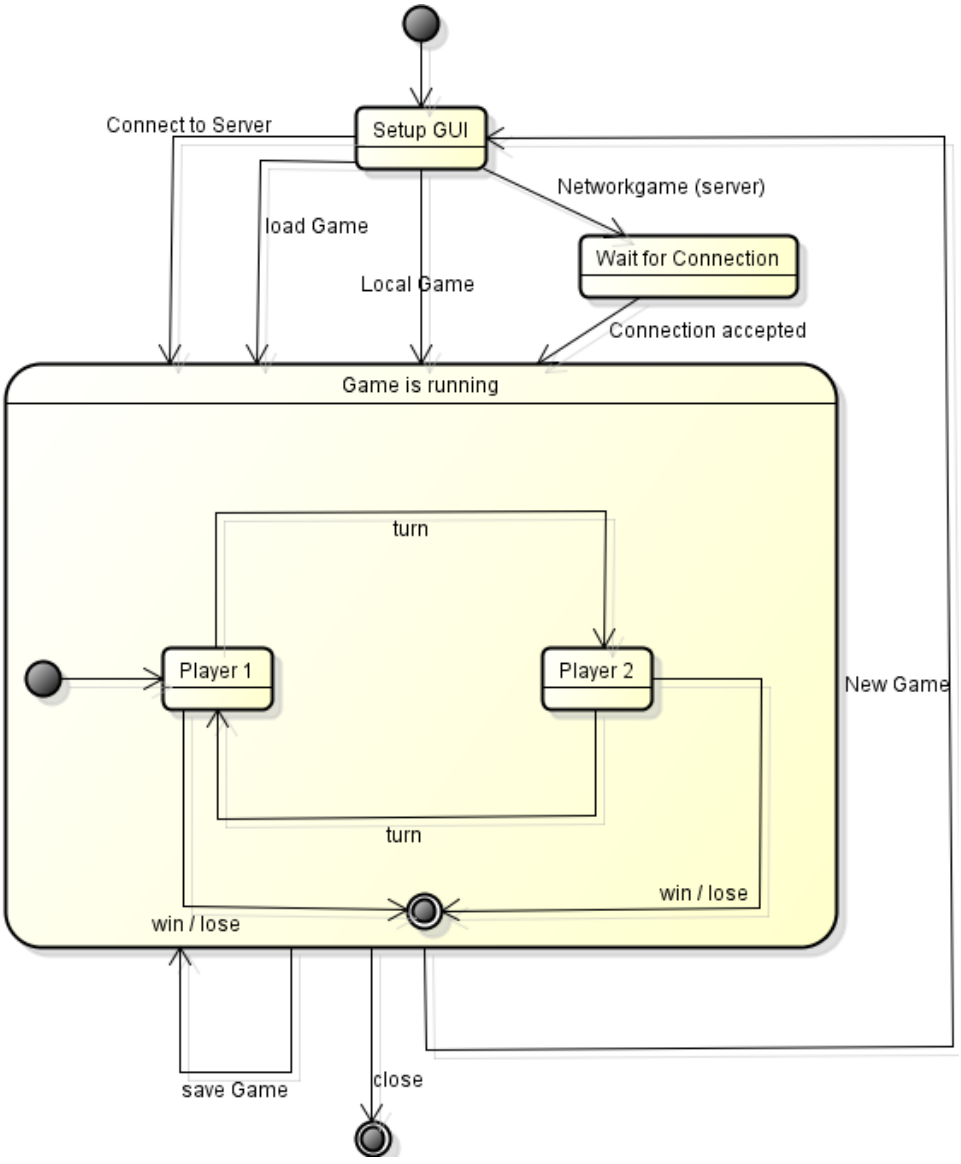


Abbildung 7: Zustandsdiagramm

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.

3.2.1.1. 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.

3.2.1.2. Game is running
Die GameController Klasse holte abwechselnd von Player 1 und Player 2 eine gültige Linien ID. Diese Abfrage findet in einem Thread statt. Hat ein Spieler das Spiel gewonnen, wird es Automatisch geresetet und beginnt von neuem.

3.2.1.3. Wait for connection
Hat das Game, die Rolle des Server wird auf eine Verbindung gewartet, bevor das Game in den Running Zustand geht.

3.3. Datensicht
Das Datenmodell ist im Wesentlichen in drei Klassen untergebracht:

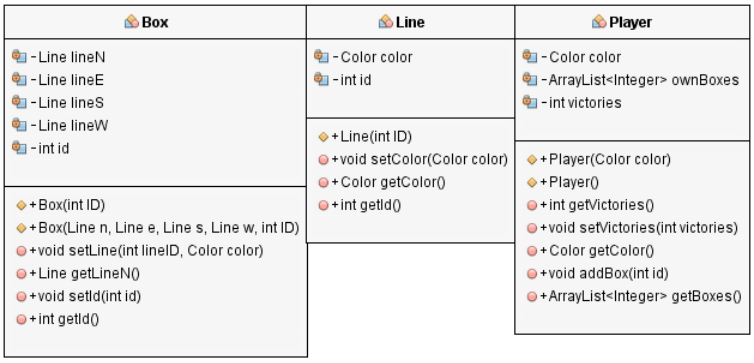


Abbildung 8: Datenmodell

Mit Hilfe von diesen drei Klassen kann die GameController Klasse ein verknüpftes Datenmodell aufbauen. Der GameController verknüpft die Zusammenhänge zwischen Player, Boxen und Lines.

Beim Speichern der Daten, werden nicht die Vorhanden Daten gespeichert, sondern jeder Spielzug der gemacht wurde. Somit ist es möglich beim Laden ein Spiel zu simulieren und so das Datenmodel wieder neu aufzubauen.

3.4. Netzwerkprotokoll
In einem Netzwerkspiel wird ein Integer wert übertragen. Das Interface IPlayer wird auch in diesem Fall angewendet. Der übertragene Integer repräsentiert die Line welche als nächstes gewählt wird.
Das Netzwerk spiel ist noch nicht fertig implementiert.

4. Erweiterungsmöglichkeiten
- Spielfeldgrösse ist dynamisch einstellbar
- Ein Spielablauf kann erneut abgespielt werden.
    - Dies ist bereits im Ladeprozess implementiert.

5. Fazit

Das Projekt ist uns ziemlich gut gelungen. Wir konnten alles ausser das Netzwerkspiel umsetzen. Eine gute Planung zu beginn der Aufgabe hat uns sehr viel geholfen. Dies ermöglichte uns die einzelnen Teilaufgaben einfach zu verteilen und den Programmcode ohne grosse Probleme zusammen zu führen. Es ist uns gelungen, uns an das geplante Klassendiagramm bis zum Schluss einzuhalten.

Viele von uns lernten den Umgang mit GitHub. Bis auf einen Zwischenfall konnten wir alle Änderungen mergen. Jedoch gab es einmal einen grösseren Konflikt, wobei Daten verloren gingen. Dies konnten wir schliesslich auch beheben.

Das Projekt hat uns allen sehr gut gefallen und wodurch wir auch viel Spass hatten. Es war eine gelungen Abwechslung zu den kurzen Übungsaufgaben.

Abbildungsverzeichnis