1. Bausteinsichten

1.1. Systemübersicht

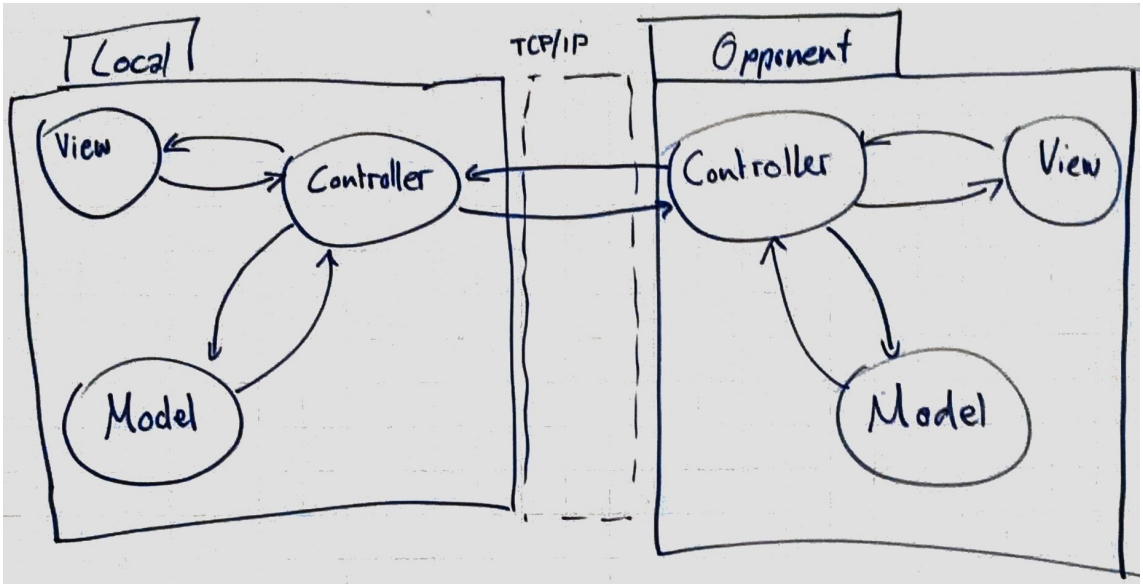

Abbildung 1 Abstraktes diagramm um den Leuten das MVC muster unseres Spieles zu erkläutern

1.2. Klassendiagramme

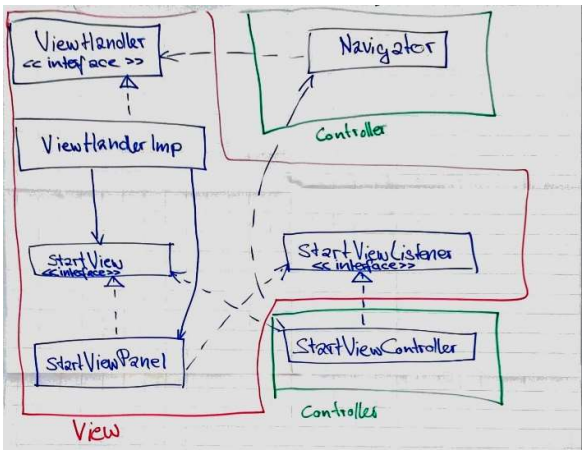

Abbildung 3 Zusammenhangs zwischen der View & den Controllern (Am Beispiel der StartView)

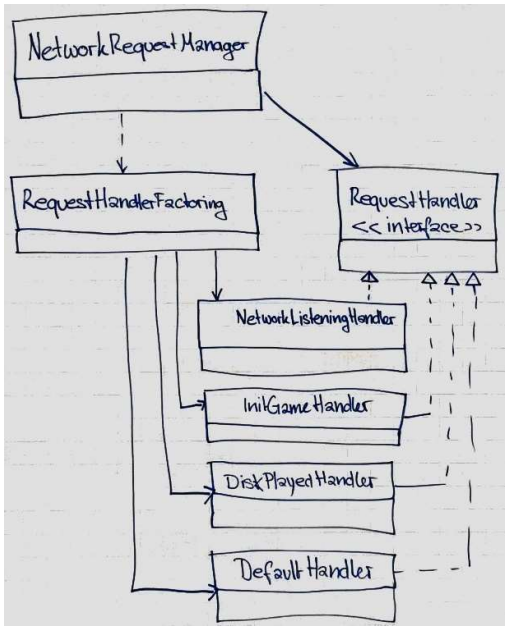

Abbildung 2 Netzwerk Klassendiagramm

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

1.3.1. GameModel
GameModel enthält den aktuellen Zustand des Spiels. Also zum Beispiel die aktuellen Spieler. Das GameModel benachrichtigt die Beobachter, falls es sich geändert hat (

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

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

2. Laufzeitsichten

2.1. Zustandsautomat
In rundenbasierten Spielen lässt sich häufig ein Zustandsautomat einsetzen. 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.

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.

2.1.2. GameRunning
Sobald die GameView startet ist das Game im Running zustand.

2.1.3. MyTurn
Spezialzustand des GameRunning zustandes. Wenn das Spiel startet und der Gegner seinen Zug beendet hat, fällt der GameRunning zustand in den MyTurn spezialzustand.

2.1.4. OpponentTurn
Beim aktiven starten eines Spiels wächselt das Spiel in den Spezialzustand OpponentTurn.
Dabei wird

2.1.5. WaitingForOpponentResponse
Bei uns wurde dieser in den OpponentTurn integriert. Da diese auf dieselbe weise Implementiert wurden.

2.1.6. 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.

2.1.7. GameOver
Sobald einer der beiden Spieler gewonnen hat, wächselt der Status auf GameOver, es wird dazu ein Dialog angezeigt. Anschliessend geht das Speil in den StartView zustand.

2.2. Sequenzdiagramme

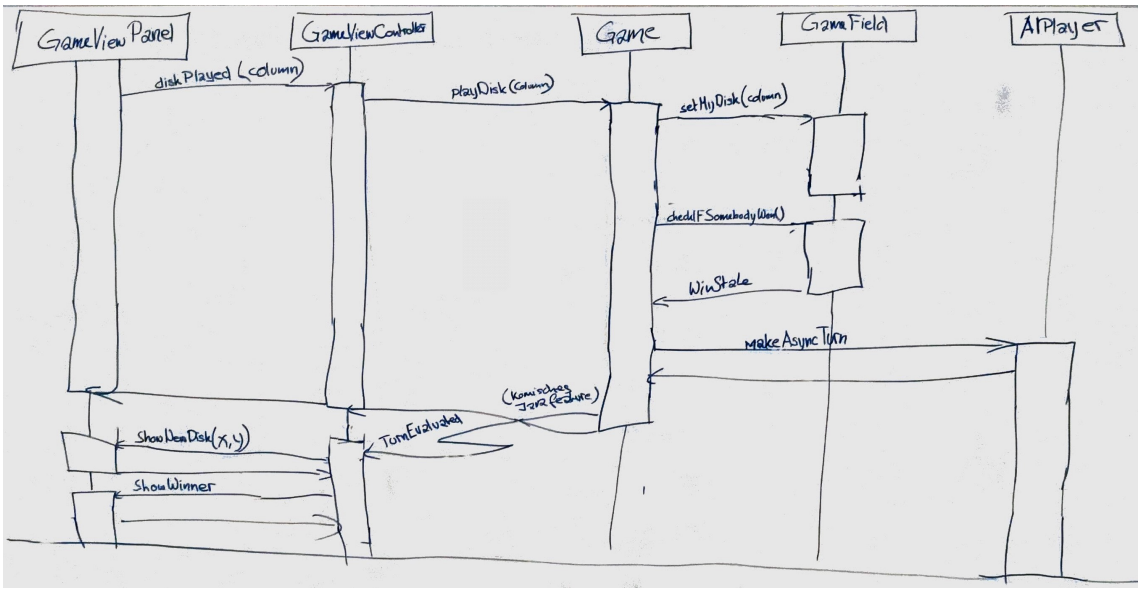

Abbildung 4 Ablauf wenn der Spieler eine Disk spielt. Bei einem Lokalen spiel.

2.2.1. PreparingGame

Fix Me! (Studenten) Sequenzdiagramm erstellen.

2.2.2. WaitingForOpponentResponse

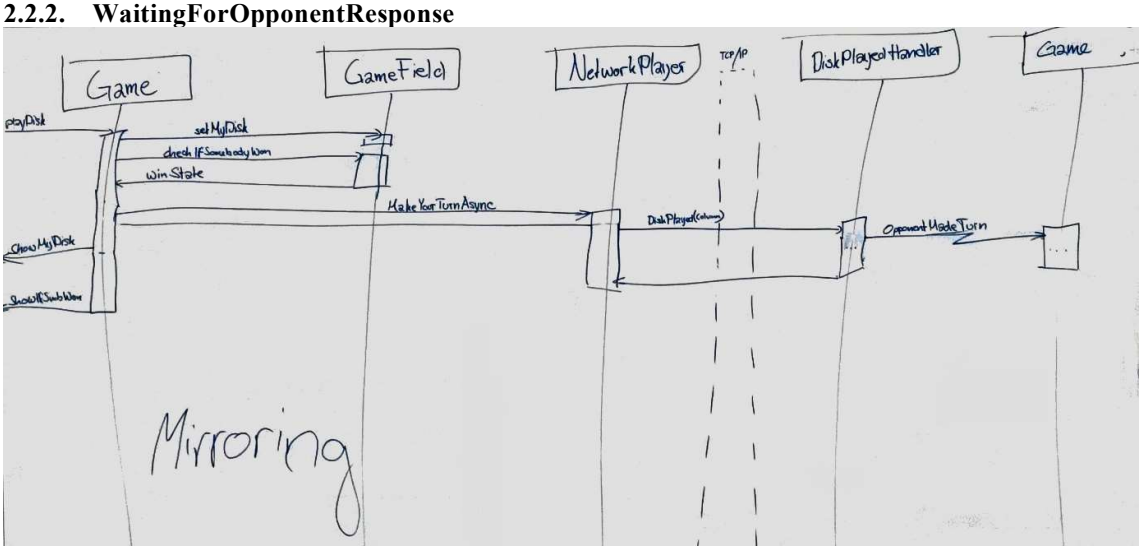

Abbildung 5 Was geschieht, wenn ein Spieler in einem Netzwerkspiel eine disk Spielt. (Bei den Punkten trifft es auf vorige Abbildung zu)

2.2.3. WaitingForInvitationResponse

Existiert nicht!

2.2.4. GameRunning

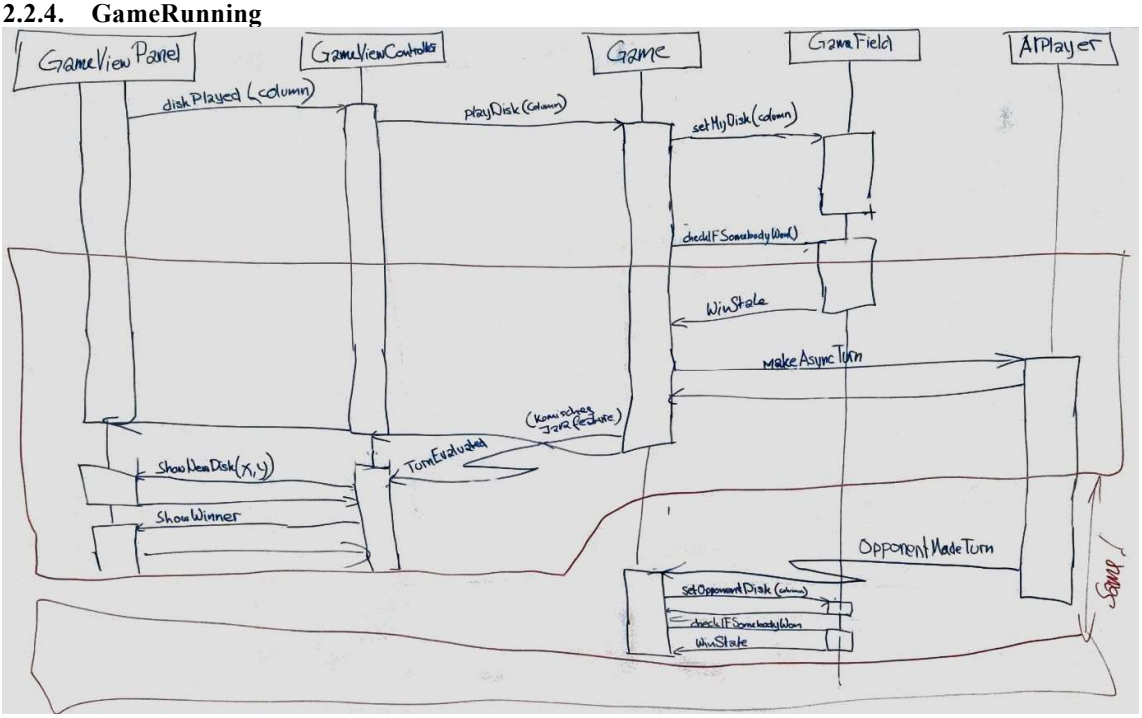

Abbildung 6 Lokales Spiel mit antwort der AI (die umrandungen bedeutet den selben ablauf)

3. Verteilungssicht

Fix Me! (Studenten) Inhalt dieses Kapitels:

- Wie läuft die Applikation im Betrieb?

Die Applikation lief während den Testphasen fliessend. Einige Fehler wurden notiert und sind unter Verbesserungen aufgeführt.

- Auf welchen Rechnern läuft die Applikation?

Die Applikation sollte auf allen Rechnern, auf denen Oracle/Open-Java läuft, einwandfrei funktionieren. Getestet wurde jedoch ausschliesslich auf Linux Systemen (Debian & ArchLinux). Kein Gewähr für andere Operating Systems (z.B. Windows, OSX, usw.).

4. Datensicht

Das Spielstand file ist nebst dem Java binary file das einzige, welches zusätzlich noch gebraucht wird. Es befindet sich im Home Verzeichnis: ~/savegame.ser

5. Netzwerkprotokoll

Fix Me! (Studenten) Netzwerkprotokoll beschreiben:

- Wie sieht die Kommunikation über das Netzwerk aus?
- Gibt es ein Protokoll?

6. Erweiterungsmöglichkeiten

Folgende Erweiterungen sind für das Projekt denkbar:

- Höhere Intelligenz des AI-Player
- Bei Netzwerkspiel Spielfeldgrösse bestimmen
- Gegenspieler fragen ob er dem Spiel beitreten möchte
- Gegenspieler über das Verlassen des Spiels informieren.