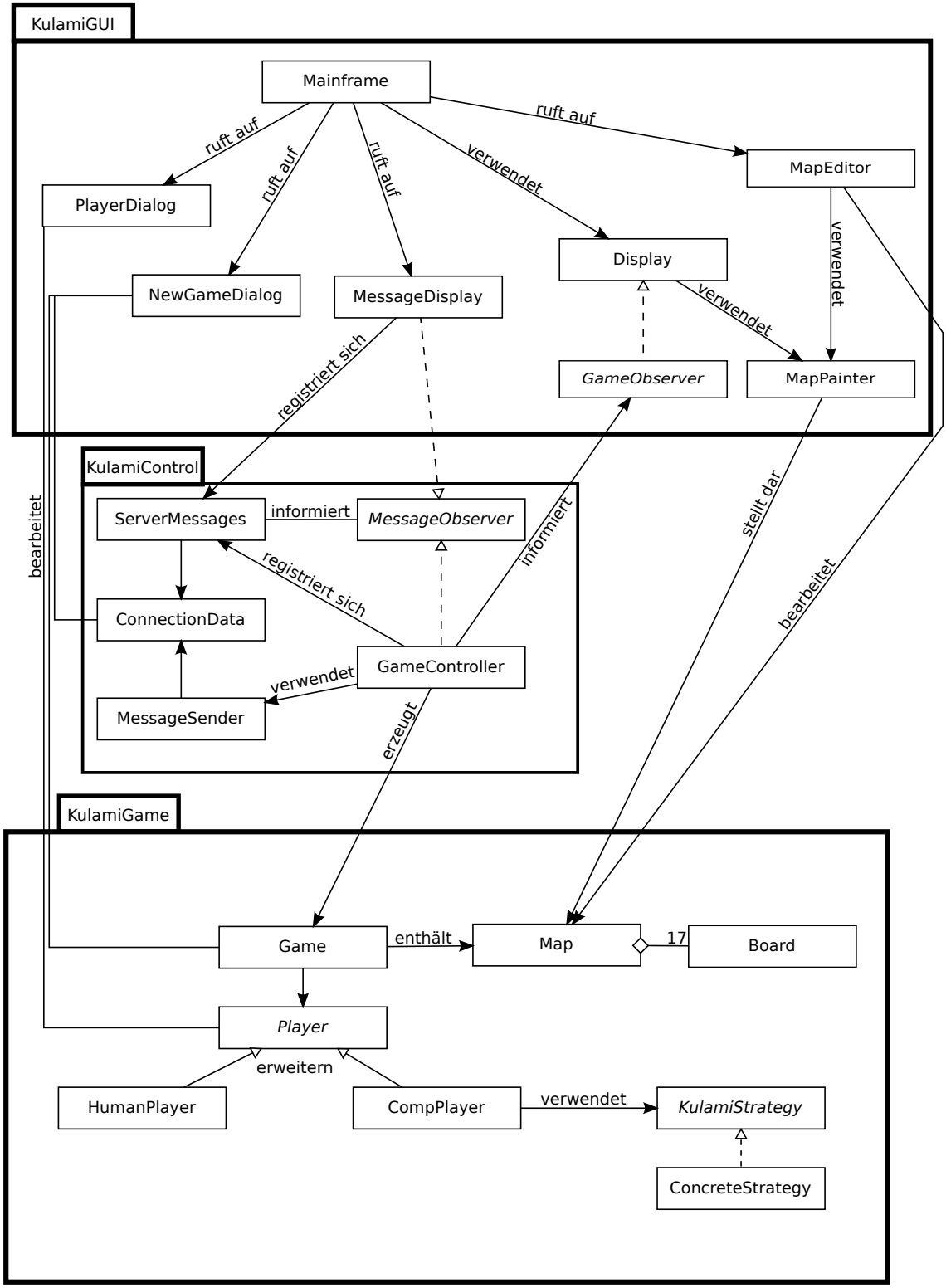
Wintersemester 2013/2014
JAVA-Programmierpraktikum 1580
Thema: Kulami (Entwurf)

Gordon Martin
Matr.-Nr. 8038813
gordon.martin@fernuni-hagen.de

17.11.2013

1 Klassendiagramm



2 Funktionalität der Klassen

Der Übersichtlichkeit halber ist das Programm in drei Pakete aufgeteilt: KulamiGame, KulamiControl und KulamiGUI.

2.1 KulamiGame

Die Klassen in diesem Paket repräsentieren ein Kulamispiel mit einem Spielfeld und einem Spieler.

2.1.1 Game

`Game` enthält das aktuelle Spiel und den Spielstand. Dazu enthält `Game` das Spielfeld, die aktuelle Belegung des Spielfeldes und Informationen zu den Spielern. `Game` enhält Methoden, die Informationen zum Spielstand liefern:

- letzter Zug
- vorletzter Zug
- zulässige Felder für den aktuellen Zug
- Plattenbesitz für jeden Spieler
- Punktestand für jeden Spieler

Außerdem enthält `Game` Methoden, um den Zustand des Spiels zu manipulieren und einen neuen Spielstand zu erzeugen:

- platziere Murmel
- breche Spiel ab

2.1.2 Player

`Player` ist eine abstrakte Klasse, die einen Spieler repräsentiert. Ein `Player` kann Murmeln platzieren und das Spiel abbrechen.

2.1.3 HumanPlayer

`HumanPlayer` repräsentiert einen menschlichen Spieler. Ein menschlicher Spieler kann Murmeln platzieren und das Spiel abbrechen. Ein `HumanPlayer` hat einen Namen.

2.1.4 CompPlayer

`CompPlayer` wird im KI-Modus verwendet. Ein `CompPlayer` ist ein automatischer Spieler, der selbständig Murmeln platziert. Ein `CompPlayer` bricht das Spiel nicht ab. Zur Auswahl des nächsten Zuges benötigt ein `CompPlayer` ein Objekt, das `KulamiStrategy` implementiert.

2.1.5 KulamiStrategy

`KulamiStrategy` ist ein Interface, das eine Methode fordert, die gegeben eines Spielstands den nächsten Zug liefert. Eine `KulamiStrategy` wird parametrisiert mit einer Stufe von 1 bis 10.

2.1.6 ConcreteStrategy

`ConcreteStrategy` ist eine Implementierung von `KulamiStrategy`. Eine `ConcreteStrategy` liefert einen konkreten Algorithmus zur Auswahl des nächsten Zuges. Es kann mehrere Implementierungen `ConcreteStrategy1`, `ConcreteStrategy2`, usw. geben, die zur Laufzeit ausgetauscht werden können.

2.1.7 Map

Die Klasse `Map` repräsentiert ein Spielfeld bestehend aus 17 Platten. `Map` bietet Methoden zum Abfragen der Plattenkonfiguration und der Belegung sowie zum Platzieren einer Murmel an.

2.1.8 Board

Ein `Board` repräsentiert eine der 17 Platten eines Spielfeldes.

2.2 KulamiControl

KulamiControl enthält Klassen, die für den Ablauf des Spiels und die Kommunikation mit dem Spieleserver verantwortlich sind.

2.2.1 GameController

Der `GameController` verwaltet den Ablauf des Spiels. Er speichert Verbindungsdaten in einem `ConnectionData`-Objekt und erzeugt ein `ServerMessages`-Objekt zum Empfang von Servernachrichten und ein `MessageSender`-Objekt zum Versand von Nachrichten an den Server. Der `GameController` implementiert das `MessageObserver`-Interface und registriert sich beim `ServerMessages`-Objekt als Nachrichtenempfänger. Der `GameController` erzeugt außerdem ein `Game`-Objekt, in dem das aktuelle Spielrepräsentiert wird ist.

2.2.2 ServerMessages

`ServerMessages` empfängt in einer Endlosschleife Nachrichten vom Kulamiserver. Die Verbindungsdaten sind in einem `ConnectionData`-Objekt enthalten. Um den Programmablauf nicht zu blockieren, läuft `ServerMessages` in einem eigenen Thread. Wenn eine neue Nachricht eingeht, informiert `ServerMessages` alle Objekte, die sich als `MessageObserver` registriert haben.

2.2.3 MessageSender

`MessageSender` enthält Methoden zum Versenden von Nachrichten an den Kulamiserver. Die Verbindungsdaten sind in einem `ConnectionData`-Objekt enhalten.

2.2.4 ConnectionData

`ConnectionData` enthält Verbindungsdaten zum Kulamiserver.

2.2.5 MessageObserver

`MessageObserver` ist ein Interface, das von allen Klassen implementiert werden muss, die von `ServerMessages` über neue Servernachrichten informiert werden wollen.

2.3 KulamiGUI

KulamiGUI enthält alle Klassen, die mit der Darstellung des Spiels zu tun haben.

2.3.1 Mainframe

`Mainframe` ist die Hauptklasse für die GUI, die alle anderen Elemente enthält. Der `Mainframe` enthält ein Spieldisplay, ruft Dialoge und den `MapEditor` auf und startet den `GameController`.

2.3.2 Display

Im `Display` wird der aktuelle Spielstand angezeigt. `Display` verwendet `MapPainter`, um das aktuelle Spielfeld zu zeichnen. `Display` implementiert `GameObserver`, um bei Änderungen des Spielstands informiert zu werden und sich zu aktualisieren.

2.3.3 MapPainter

`MapPainter` ist eine Hilfsklasse, die ein Spielfeld zeichnen kann.

2.3.4 GameObserver

`GameObserver` ist ein Interface, das von Klassen implementiert wird, die über Änderungen des Spielstands informiert werden wollen.

2.3.5 MapEditor

Der `MapEditor` ist ein graphischer Editor, in dem Spielfelder erstellt und bearbeitet werden können.

2.3.6 PlayerDialog

Im `PlayerDialog` kann ein neuer Spieler erzeugt werden.

2.3.7 NewGameDialog

Mit dem `NewGameDialog` wird ein neues Spiel gestartet. Für ein neues Spiel müssen eine Serverbindung (`ConnectionData`), ein Spieler (`Player`) und ein Spielfeld (`Map`) zur Verfügung stehen. Im `NewGameDialog` gibt der Benutzer die Verbindungsdaten zum Kulamiserver an und wählt ein Spielfeld aus.

2.3.8 MessageDisplay

Im `MessageDisplay` werden Nachrichten für den Spieler angezeigt. Dies können Mitteilungen des gegnerischen Spielers oder für den Spieler relevante Servernachrichten sein. Um über Servernachrichten informiert zu werden, registriert sich `MessageDisplay` bei `ServerMessages` als `MessageObserver`.

3 Beziehungen zwischen den Klassen

Das Programm wird im `Mainframe` gestartet. Der `Mainframe` startet den `GameController`. Der `GameController` hat Assoziation zu allen drei Bereichen des Programms. Damit ist `GameController` die eigentliche Hauptklasse, die die verschiedenen Komponenten miteinander verbindet. Insbesondere verwaltet der `GameController` ein Objekt mit den Serververbindungsdaten und startet den Empfang vom und den Versand von Nachrichten an den Server. Der `GameController` hat außerdem ein Modell des Spielzustands und informiert bei Änderungen des Spielzustands alle registrierten `GameObserver`, also das `Display`-Objekt, welches für die Darstellung des Spiels zuständig ist.

Das Spielzustandsmodell besteht aus der Klasse `Game` und den mit ihr verbundenen Klassen `Map` und `Player`. `Map` repräsentiert das Spielfeld, bestehend aus 17 `Board`s. `Game` speichert sowohl die verwendete `Map` als auch die `Map` mit der aktuellen Belegung. `Player` ist eine abstrakte Klasse, die von `HumanPlayer` und `CompPlayer` erweitert wird. Zu `CompPlayer` gehört eine `KulamiStrategy`, die als Interface verbunden ist, damit verschiedene konkrete Strategien ausprobiert werden können.

Die KulamiControl-Komponente enthält Klassen zur Kommunikation mit dem Spieleserver. `ServerMessages` empfängt Nachrichten vom Server und informiert alle registrierten `MessageObserver`. `MessageObserver` sind der `GameController`, der auf Nachrichten ggf. mit Änderungen des Spielzustands reagiert, und das `MessageDisplay`, das dem Benutzer relevante Nachrichten (z.B. Chatnachrichten) anzeigt. Eine Debuggingkonsole könnte auf die gleiche Art angeschlossen werden. Der `MessageSender` sendet Nachrichten an den Server.

Die Benutzeroberfläche ist im `Mainframe` enthalten. Die Klasse `Display` ist dafür zuständig, das aktuelle Spielfeld darzustellen. Für das Zeichnen der Map wird eine eigene Klasse `MapPainter` verwendet. `PlayerDialog` ist zuständig für die Erstellung eines Spielerprofils. Der `MapEditor` ist ein Editor zur Erstellung und Bearbeitung von Spielfeldern und verwendet ebenfalls `MapPainter` für die Darstellung des Spielfeldes.

Bevor ein Spiel gestartet werden kann, muss der Benutzer mit dem `PlayerDialog` einen Spieler anmelden. Dann kann der Benutzer mit dem `NewGameDialog` die Verbindungsda-

ten zu einem Kulamiserver angeben und den Client mit dem Server verbinden. Falls der Benutzer als erster Spieler einem Spiel beitritt, muss er ein Spielfeld auswählen.