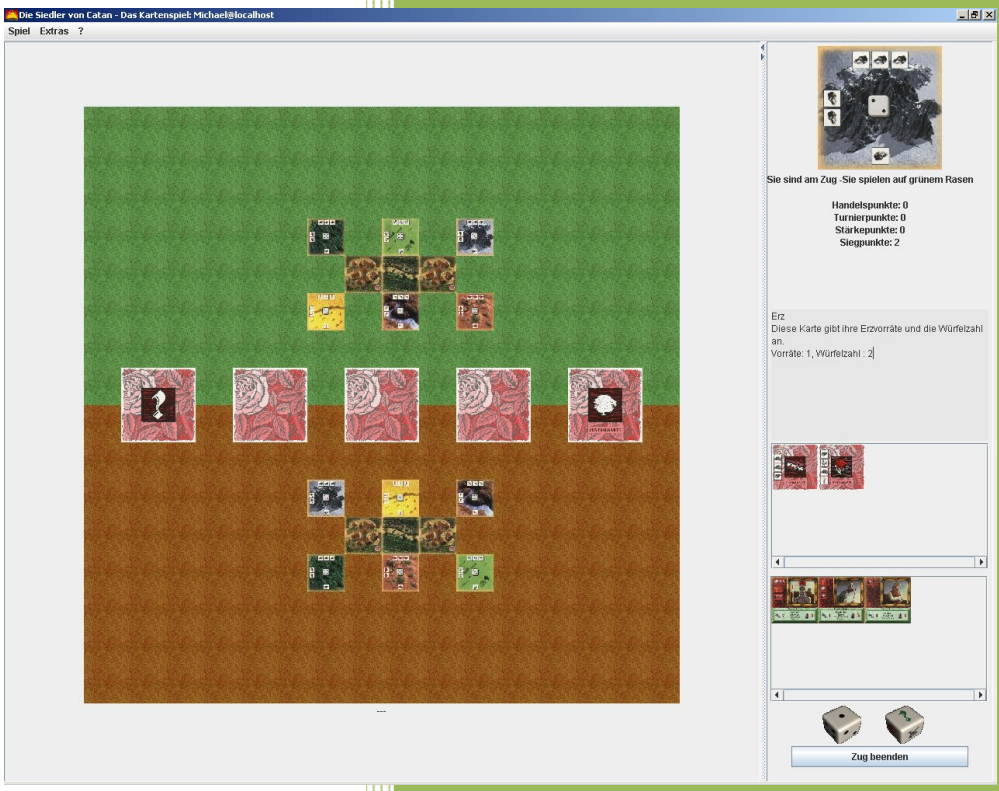
2008

Softwareprojekt



Die Siedler von Catan - Das Kartenspiel

als Verteilte Anwendung mittels Java und RMI

Marcus Zelend

Michael Körner

Danny Christl

18.03.08

Inhalt

Abbildungsverzeichnis

1. Einleitung und Zielsetzung

Im Rahmen des Softwareprojektes stellten wir uns die Aufgabe das Kartenspiel „Siedler von Catan“ für den PC umzusetzen. Das Spiel sollte so umgesetzt werden, dass es auf möglichst vielen Systemen und zwischen mehreren Personen nutzbar ist. Infolge dieser Anforderungen entschieden wir uns für eine Realisierung in JAVA unter Verwendung von RMI.

In der Lehrveranstaltung „Verteilte Systeme“ wurde uns ein Entwurfsmuster zur Verfügung gestellt, welches wir im Laufe der Programmierarbeiten weiterverwendeten.

Hauptziel unser Programmierarbeiten war somit eine möglichst effiziente Nutzung der gegebenen Schnittstellen, sowie eine klare Strukturierung der neu hinzugekommenen Klassen zur Entwicklung eines Spiels, welches weiterverwendet und durch Funktionalitäten erweitert werden kann.

2. Vorüberlegungen und Spielkonzept

Aufbauend auf dem RMI-Konzept ist eine Reihe von möglichen Implementierungen möglich. Deshalb haben wir uns für „Die Siedler von Catan- Das Kartenspiel“ entschieden, welches dafür bestens geeignet schien. Dieses Spielkonzept ermöglicht eine hohe Interaktion der grafischen Oberfläche mit dem Spieler, welches wiederum zahlreiche Möglichkeiten bot den Zugriff auf entfernte Objekte und den damit verbundenen Transport über das Netzwerk zu ermöglichen.

Das Kartenspiel Siedler von Catan ist ein Spiel für zwei Spieler, welche in der Regel nacheinander am Zug sind. Zu Beginn des Spieles besitzt jeder Spieler ein Fürstentum, welches grundlegend aus einer Stadt, zwei angebauten Straßen und sechs Ressourcen besteht. Die Ressourcen haben im Verlauf des Spieles den Zweck sich weiter Anbauten für sein Fürstentum zu leisten. Des Weiteren gehörten jedem Spieler drei Handkarten, deren Anzahl sich jedoch im Verlauf des Spiels erhöhen kann.

Um in das Spielgeschehen einzugreifen hat der Spieler nun etliche Möglichkeiten. Zum Einen kann er die Handkarten spielen, die je nach Typ Einfluss auf Ressourcen, Gebäude vom Spieler selber oder auch vom Gegner haben.

Zum Anderen existieren im Spiel zwei Würfel, die ebenfalls unterschiedliche Ereignisse auslösen (Ereigniswürfel) oder Ressourcen erhöhen (Zahlenwürfel). Beim Würfeln werden jeweils beide Würfel gleichzeitig benutzt und die jeweiligen Ereignisse ausgeführt.

Im Spiel existiert für jeden Spieler eine Reihe von Punkten, die in der folgenden Tabelle verdeutlicht werden:

Punkte	**Einfluss/Auswirkung**
Siegpunkte	- Einfluss auf Spielgewinn (12 Punkte führen zum Spielgewinn) - abhängig von Gebäuden und Anbauten
Stärkepunkte	- Auswirkungen auf Besitz der Rittermacht (1 Siegpunkt) - abhängig von Anbauten (Ritter)
Turnierpunkte	- Auswirkung auf das Ereignis „Turnier“ - abhängig von Anbauten (Ritter)
Handelspunkte	- Auswirkung auf den Besitz der Mühle (1 Siegpunkt) - abhängig von Anbauten (z.B. Markt)

Tabelle 1: Übersicht über die Punkte des Spielers

Das äußerst komplexe Regelwerk komplett zu erklären würde die Grenzen dieser Abhandlung übersteigen und ist deshalb extrem gekürzt. Jedoch gibt es zahlreiche externe Quellen[1], die näher auf den Spielverlauf eingehen.

Ziel des Spieles ist es durch taktische Überlegungen und Ausbau des Fürstentums möglichst schnell an die geforderte Anzahl von 12 Siegpunkten zu kommen und somit das Spiel für sich zu entscheiden.

[1] http://www.profeasy.de/Kartenspiel-Einfuehrung/html/kartenspiel01.html

3. Gesamtkonzept und grundsätzlicher Aufbau:

Die Implementierung des Spiels Siedler wurde auf Basis des Spiels Reversi entwickelt, welches uns in der Lehrveranstaltung „Verteilte Systeme“ zur Weiterverarbeitung zur Verfügung gestellt wurde. Im folgenden wird diese Vorlage vereinfachend als Beispielprojekt beschrieben.

3.1 Aufbau des Beispielprojektes:

Im vorgegebenen Beispielprojekt wurde das Netzwerkspiel Reversi mittels RMI umgesetzt, einer Übertragungstechnik, durch die Methoden von Objekten in einer entfernten virtuellen Maschine aufgerufen werden. Zur besseren Weiterverarbeitung wurde das Projekt in Bereiche untergliedert (siehe **Fehler! Verweisquelle konnte nicht gefunden werden.**). Diese Aufteilung erfolgte nicht nur logisch durch das Klassendesign, sondern auch physikalisch, wodurch Client und Server auf verschiedenen Rechnern laufen können, oder zumindest in verschiedenen Prozessen realisiert werden. Durch die Abtrennung des Kommunikationslayers ist es möglich das Projekt ohne größeren Aufwand auf einen anderen Übertragungsmechanismus, wie zum Beispiel TCP/IP, zu portieren. Genauso einfach kann der Client und Server, also den spielspezifischen Quellcode, ausgetauscht werden, was wir sogleich zur Umsetzung des Softwareprojektes genutzt haben. Im Beispielprojekt wurde je ein Objekt zur Übermittlung der Daten vom Client zum Server und umgekehrt erzeugt. Dieses Objekt beinhaltete nun entweder den jeweiligen Spielzug des Clients oder die aktuellen Daten des Servers.

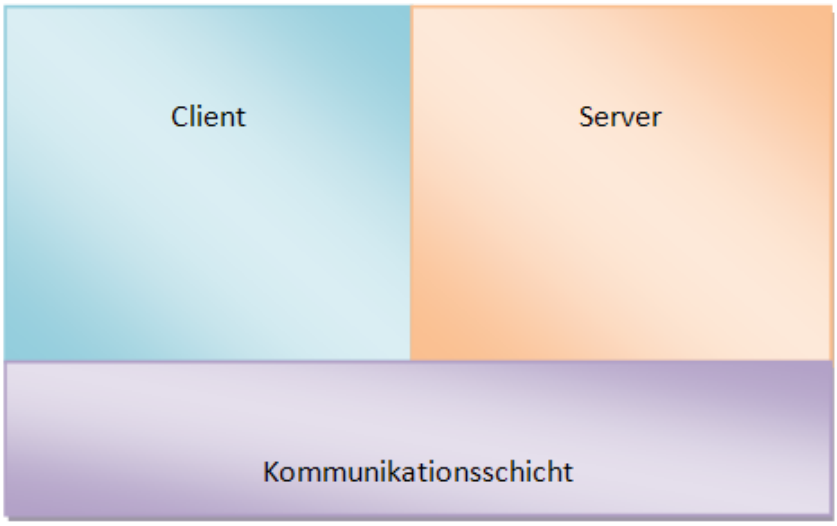


Abbildung 1: Gliederung des Beispielprojektes

3.2 Umsetzung des „Model View Controller“- Musters auf dem Client

Als wichtige Grundlage zur Weiterverarbeitung des Quellcodes, wurde die klare Gliederung des Beispielprojektes beibehalten. Es wurde Wert darauf gelegt, eine strikte Trennung der Aufgaben vorzunehmen. Der Server ist dabei verantwortlich für die eigentlichen Spieldaten, sowie die gesamte Spiellogik und bildet dadurch das Modell. Die Kartenstapel und die Figuren werden demzufolge nicht beim Spieler selbst auf dem Client verwaltet, sondern auf dem Server. Die Aufgabe des Clients besteht infolgedessen lediglich in der Repräsentation der empfangenen Daten sowie in der Bedienung der Spielelemente.

Die Steuerung der Ausgabeelemente, wie das Anzeigen der aktuellen Handkarten oder das Darstellen des Würfelereignisses, wird in unserem Projekt durch den SettlerController sowie den GameGuiController ausgeführt. Der Controller verwendet dabei beispielsweise ein Objekt des Spielfeldes und führt darauf die Darstellungsanweisungen aus. Auch Daten, die der Spieler in Form von Spielzügen eingibt, werden an die Controller weitergeleitet und von dort aus an den Server gesendet.

Die Darstellungsobjekte SettlerPanel, MenuPanel sowie SettlerFrame dienen somit nur der Präsentation, wohingegen die Controller die Steuerung der Elemente vornimmt (Abbildung 2: Zusammenhang View/Controller).

Durch diese Trennung von Modell, Präsentation und Steuerung, wird das Architekturmuster des Model-View-Controller erfüllt, was dazu dienen soll, ein flexibles Programmdesign zur späteren Änderung oder Erweiterung zu erleichtern.

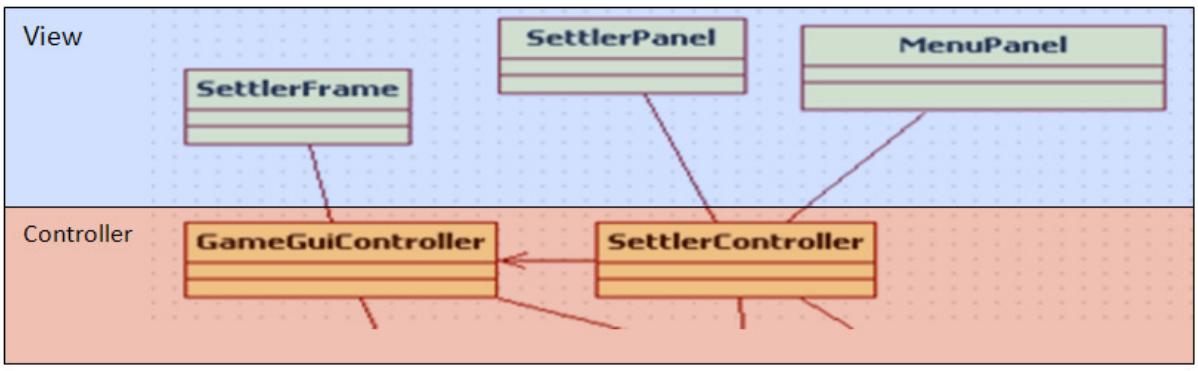


Abbildung 2: Zusammenhang View/Controller

3.3 Verwendung von Interfaces am Beispiel Player-Game auf dem Server

Die zentrale Struktur des Spieles besteht aus einem Spieler (Player) und einem Spiel (Game) welche in Beziehung zueinander stehen. Mehrere Spieler nehmen demzufolge an einem Spiel teil, wobei jeder Spieler an maximal einem Spiel beteiligt sein darf. Unabhängig von der jeweiligen Implementierung tritt dieses Muster bei Reversi als auch bei Siedler auf. Die Klasse Game bildet in diesem Zusammenhang die Basisklasse für die eigentliche Implementierung, auf welche im nächsten Kapitel näher eingegangen werden wird.

Diese Beziehung soll nun dazu dienen, das Schnittstellenkonzept, welches im gesamten Projekt verwendet wird, näher zu erläutern.

Die Zustände von Player und Game können sich im Laufe des Spieles verändern. Ein einfaches Beispiel wäre, dass der Spieler 1 am Zug sein kann oder nicht. Genauso kann sich das Spiel selbst ändern, indem Karten vom Spielfeld genommen werden. Um diesen Informationsaustausch zu gewährleisten, implementiert der Player das Interface GameListener (Abbildung 3: Verwendung von Interfaces). Jeder GameListener jedes Spielers ist im Game selbst registriert und erhält bei Modifikationen die Änderungsinformationen. Sicher wäre diese Funktionalität auch ohne Interfaces realisierbar gewesen, jedoch bietet dieses Vorgehen die Möglichkeit, einfache weitere Funktionen zu implementieren und Quellcode auszutauschen. So kann es ermöglicht werden, Dritte am Spiel teilhaben zu lassen, ohne ihnen die Möglichkeit zur Intervention zu bieten. In diesem Fall würde Player ein weiteres Interface zur passiven Teilnahme am Spiel implementieren.

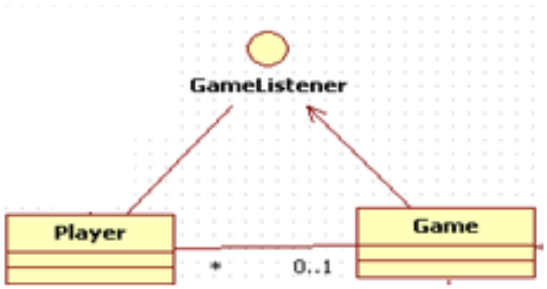


Abbildung 3: Verwendung von Interfaces

3.4 Die Kommunikationsschicht

Wie im einführenden Kapitel bereits beschrieben, wurde das Projekt mittels RMI realisiert. Sämtlich spezifische Sendefunktionalitäten sind somit in diese Schicht eingebettet. Der Client wird in dieser Netzwerkschicht durch eine GameClientBean repräsentiert, analog dazu, vertritt die PlayerBean und die GameServerBean den Server. Zur Interaktion zwischen diesen Vertretern werden Interfaces definiert, die die Kommunikation (siehe 1.3) ermöglichen . Die GameClientBean kann dadurch Aufrufe an das PlayerInterface sowie das GameServerInterface veranlassen (Abbildung 4: Aufbau der Netzwerkschicht). Umgekehrt kann der Player seiner grafischen Repräsentation Änderungsinformationen über das Interface PlayerListener zukommen lassen. Mit Hilfe des GameServerInterfaces werden Login-Daten an den Server gesendet und somit eine Referenz auf den Client geliefert. Eine Kommunikation vom GameServer zum GameClient ist im Laufe des Spiels nicht mehr notwendig, da nach dem Login sämtliche Informationen über die Playerobjekte verteilt werden. Die Kommunikation zwischen Netzwerkschicht und Client bzw. Server könnte in diesem Schritt fest durch Referenzen umgesetzt werden. Allerdings würde dieses Vorgehen ein einfaches Ersetzen des Kommunikationslayers verbieten. Die Realisierung kann dadurch wiederum nur durch Interfaces erfolgen.

In der Realisierung implementiert die GameClientBean verschiedene Interfaces, die nun vom Client unterschiedlich genutzt werden (Abbildung 5: Verbindung der Netzwerkschicht). So wird der PlayerService beispielsweise verwendet um Spielzüge an den Server weiterzuleiten. Analog dazu implementiert der SettlerController ein Interface um Änderungen vom Server zu erhalten. Eine Vergleichbare Architektur lässt sich auf der Serverseite beobachten, wobei hier ein PlayerService und ein PlayerListenerService für den Datenaustausch verantwortlich sind.

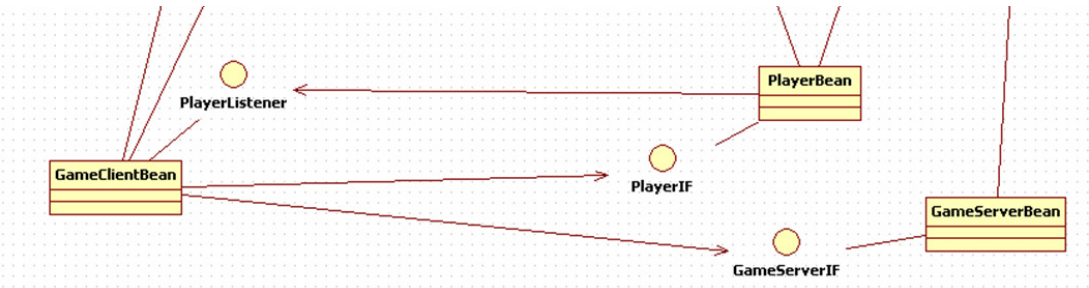


Abbildung 4: Aufbau der Netzwerkschicht

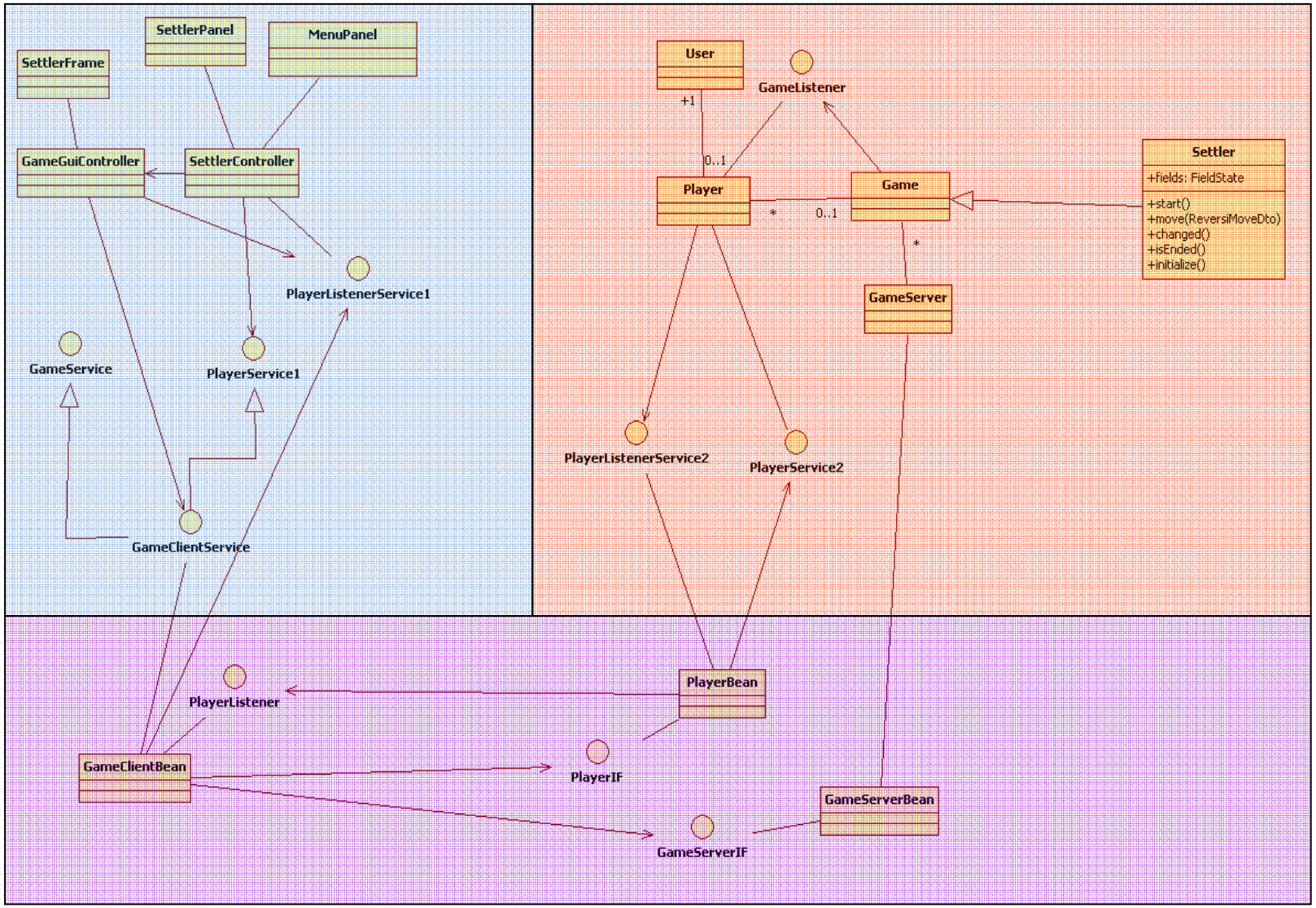


Abbildung 5: Verbindung der Netzwerkschicht

3.5 Der Austausch mittels Transferobjekten

Grundsätzlich werden Daten zwischen Client und Server mittels sogenannten Data Transfer Objects (kurz Dto) ausgetauscht, wobei unterschieden werden muss, in welche Richtung die Daten gesendet werden. Daten vom Client zum Server sind während des Spieles hauptsächlich Spielzüge, die von Spielern ausgeführt werden. Informationen, die zurückgesendet werden, beinhalten aktuelle Aufforderungen oder aktuelle Daten des Servers. Im vorgegebenen Beispielprojekt übernahmen diese Funktion die GameDto sowie die MoveDto. Im Laufe unserer Programmierarbeiten stellte sich jedoch heraus, dass eine Weiterverwendung dieser Objekte infolge der inhomogenen Anwendungsfälle unmöglich umzusetzen war (Abbildung 6: Anwendungsfälle während des Spieles). Aufgrund des unterschiedlichen Gebrauchs der Dtos entschieden wir uns dazu, eine Vererbungshierarchie aufzubauen (Abbildung 7: Vererbungshierarchie der MoveDto sowie Abbildung 8: Vererbungshierarchie des GameDtos). Sendet hier zum Beispiel der Server an den Client Informationen über ein Würfelereignis, so beinhaltet das gesendete Objekt in jedem Fall die gewürfelte Zahl, jedoch durch die Vererbung auch Daten zur Auswertung des Ereigniswürfels. Dies kann dazu führen, dass dem Gegner Rohstoffe entzogen oder andere Ereignisse ausgeführt werden, womit weitere Dtos zur Gestaltung des Spielflusses zum Einsatz kommen.

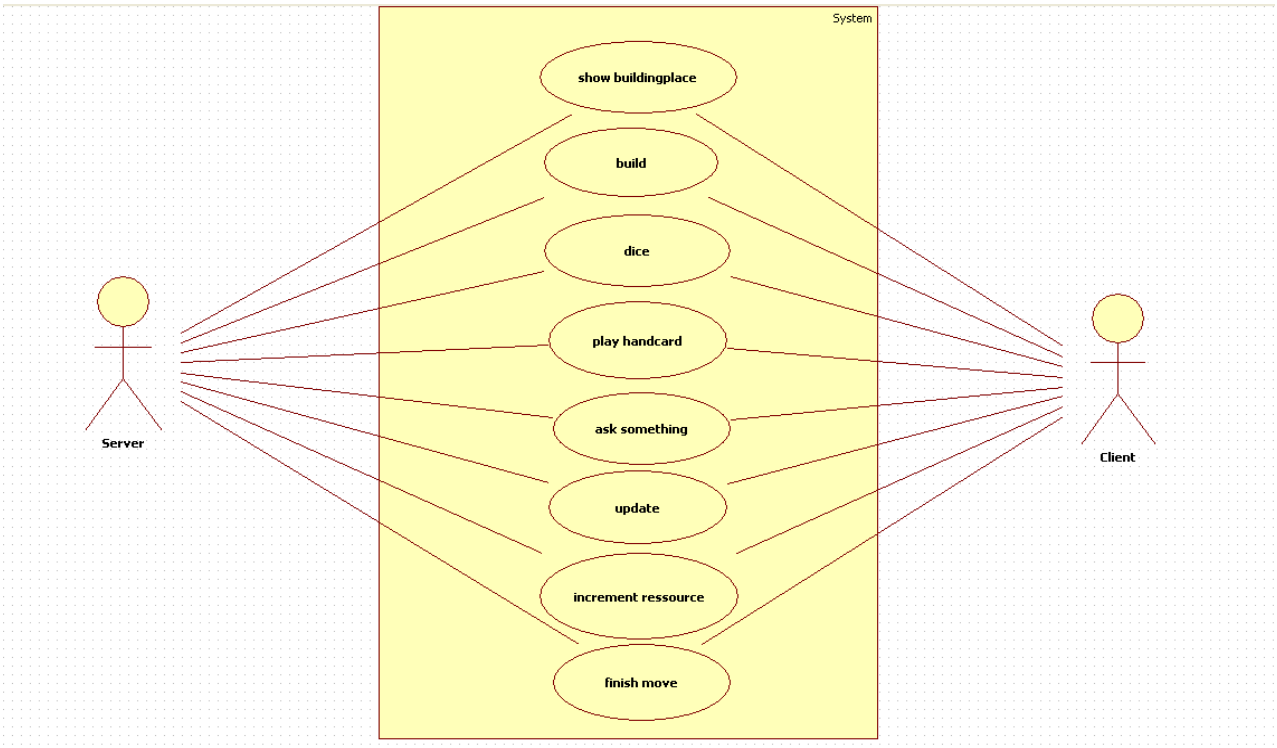


Abbildung 6: Anwendungsfälle während des Spieles

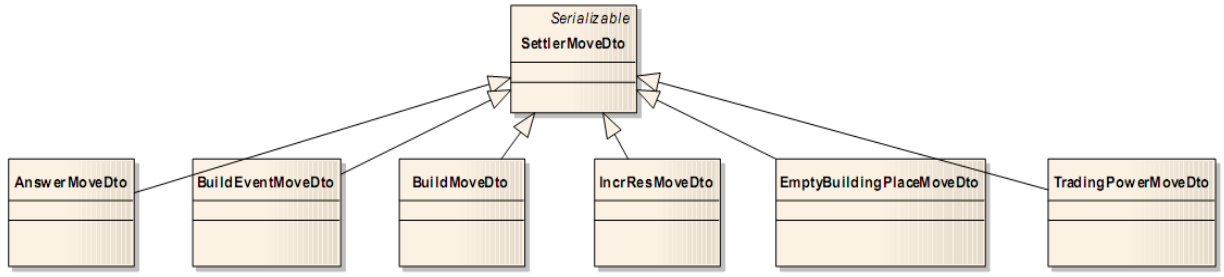


Abbildung 7: Vererbungshierarchie der MoveDto

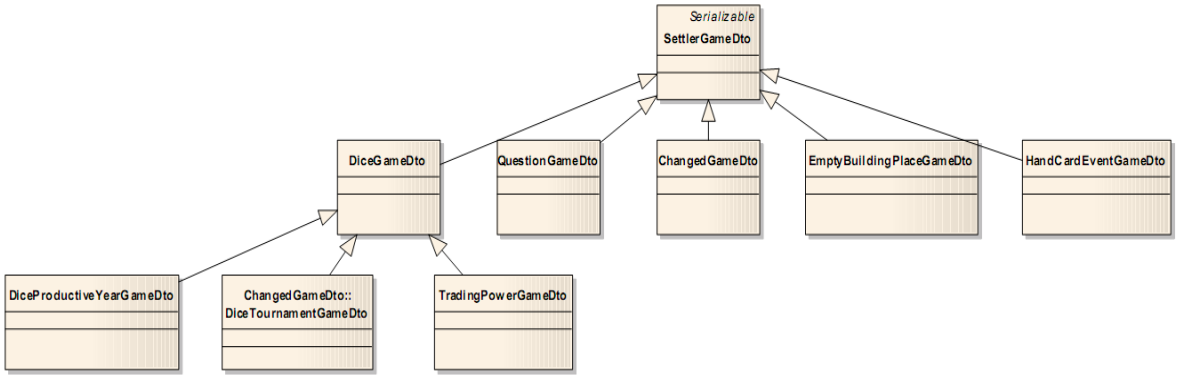
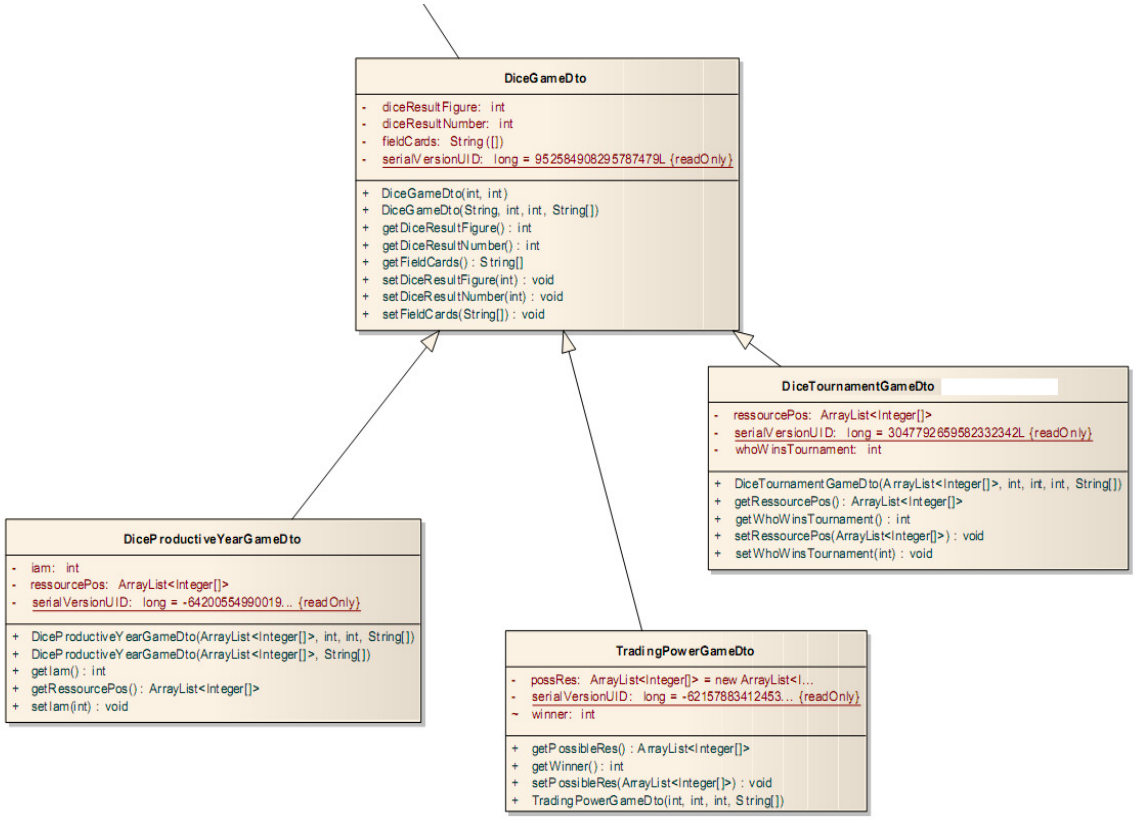


Abbildung 8: Vererbungshierarchie des GameDtos



4. Die grafische Oberfläche

Die grafische Oberfläche wurde mittels Swing erstellt und besteht im für den User sichtbaren Teil aus zwei JPanels – den Kartenbereich und den Informationsbereich -, die durch eine JSplitPane geteilt sind. Das Layout beider JPanels ist das GridBagLayout, welches gute Skalierbarkeit in Bezug auf die Karten und den Informationsbereich bietet. Im Hintergrund werden die Interaktionen durch Controller gesteuert. Dieser Zusammenhang ist im folgenden Bild sichtbar:

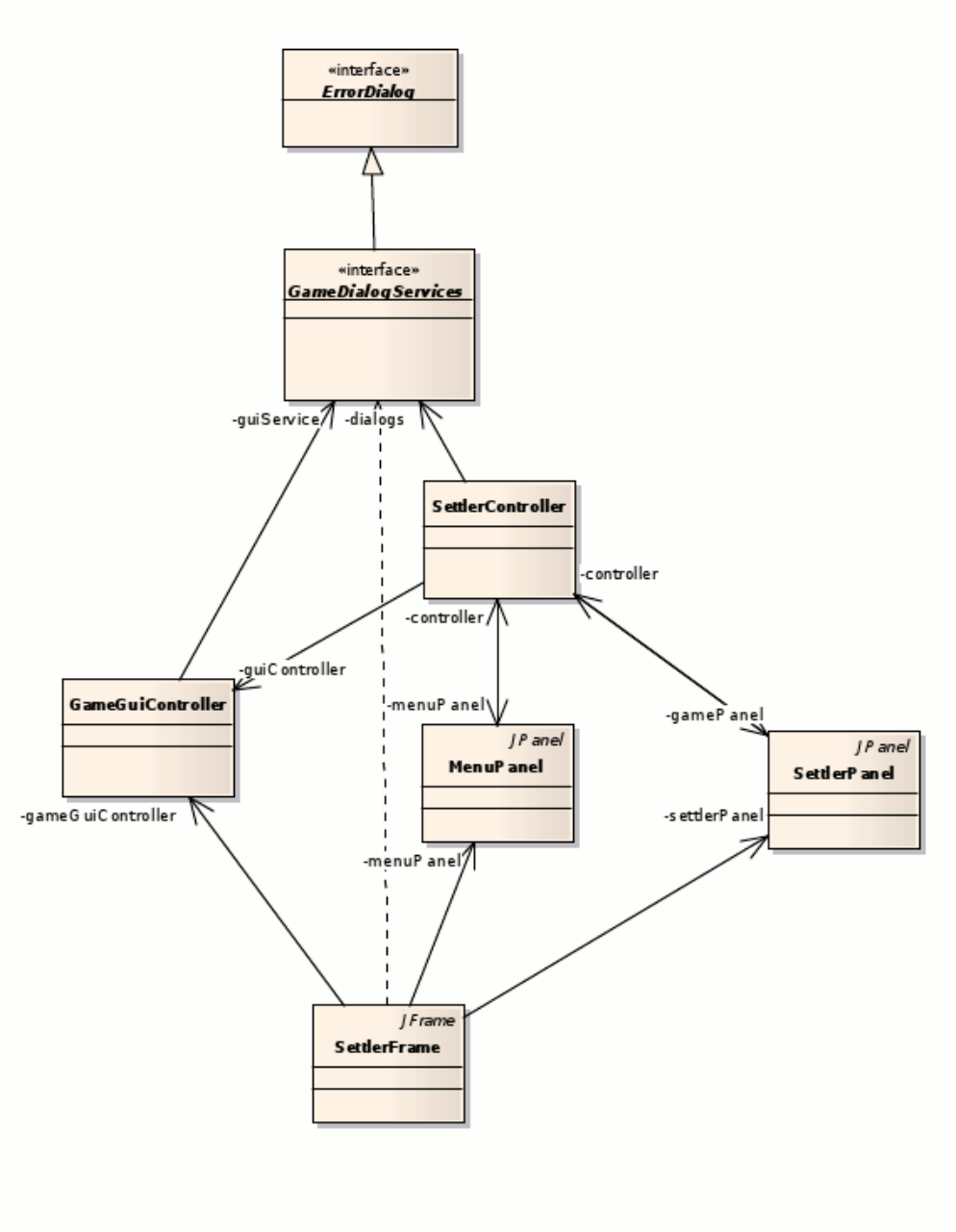


Abbildung 9: UML-Diagramm der grafischen Oberfläche

Das komplette Fenster der GUI besteht aus einem JFrame, dem SettlerFrame. Dieser JFrame dient als Top-Level-Container für die sichtbaren Elemente, wie das Spielfeld, den Informationsbereich und die Menüleiste.

Die Steuerung der GUI übernimmt zum Einen GameGuiController, welcher für die Steuerung der Menüs und der Benutzerdialoge zuständig ist. Zum Anderen gibt es noch den SettlerController, der die Steuerung auf dem Spielfeld –repräsentiert durch das SettlerPanel- und im Informationsbereich – dargestellt durch das MenuPanel- übernimmt. Hier werden zum Beispiel die MouseListener verwaltet oder die Karten auf dem Spielfeld gesetzt bzw. ausgetauscht.

Im Diagramm sieht man besonders, dass die Controller beide gegen das Interface GameDialogService programmiert wurden. Dies vereinfacht deutlich die Kommunikation der einzelnen Schichten untereinander und unterstützt somit das Konzept der Strikten Trennung der Schichten.

5. Funktionalitäten des Servers

Der Server übernimmt im Projekt die komplette Spiellogik und die Steuerung der Zustände des Spiels. Demzufolge sind die involvierten Klassen entsprechend groß. Der Server besteht zum Einen aus der Spielklasse, durch die es möglich ist weitere Spiele zu implementieren. Diese Klasse ist recht allgemein gehalten und stellt nur Grundfunktionalitäten für das Spiel bereit, wie z.B. das Starten des Spiels oder das Verwalten der Teilnehmer des Spiels. Darauf aufbauend ist es nun möglich die unterschiedlichsten Spiele auf dieser Architektur zu nutzen, da diese Funktionalität bei jedem verteilten Spiel benötigt wird. Ein weiteres Element ist die Klasse für den Spieler, welche Interaktionen des Spielers zur Verfügung stellt. Das wären unter Anderem das Teilnehmen am Spiel oder das Anfordern eines Gegners über den Server. Die Serverklasse als drittes Hauptelement verwaltet alle angemeldeten Spieler und stellt diese auch bereit. Des Weiteren stellt startet sie den Server und hört den eingestellten Port ab, um Anfragen der Clients zu beantworten.
Im nachfolgenden UML-Diagramm sind diese Zusammenhänge noch einmal verdeutlicht:

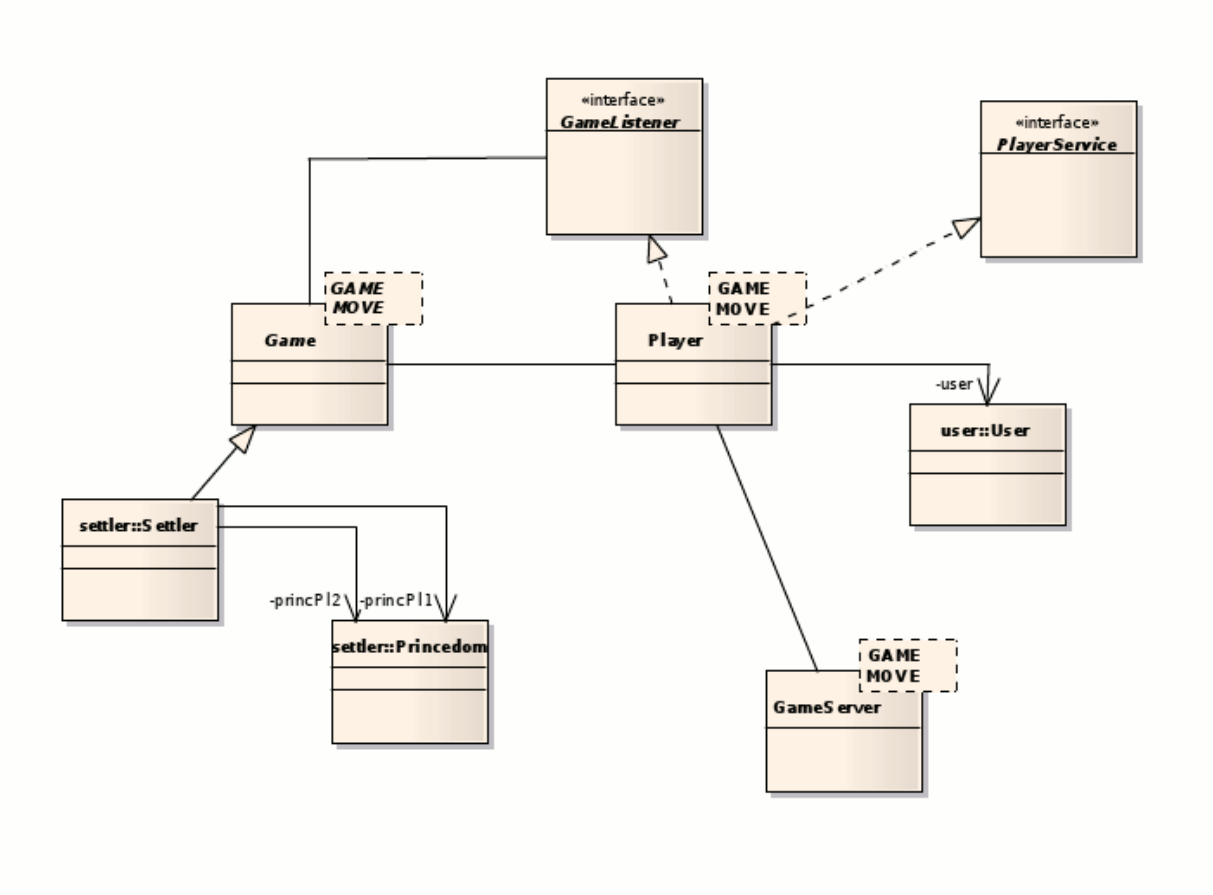


Abbildung 10: UML-Diagramm des Servers

In der Grafik sieht man ebenfalls wie das Konzept des Programmierens gegen Interfaces umgesetzt wurde, da das Spiel und auch die Spieler über dasselbe Interface implementiert sind.

Die eigentliche Spiellogik befindet sich in der Klasse Settler -welche von Game abgeleitet ist- und in der Klasse Princedom, die für das Verwalten des Fürstentums, inklusive der Handkarten des Spielers, seiner Punkte und der Karten auf dem Spielfeld zuständig ist. Aus diesem Konzept ergeben sich natürlich etliche Membervariablen und Methoden für beide Klassen, welche im folgenden Diagramm genauer dargestellt werden:

Abbildung 11: Membervariablen und Methoden des Spiels

Die Klasse Princedom stellt Methoden bereit, die für den Spielfluss benötigt werden. Das wären unter Anderem das Bereitstellen der leeren Felder für Anbauten jeglicher Art, das Erhöhen oder das Vermindern der Ressourcen, die Verwaltung aller spielrelevanten Punkte und der Figuren. Dies geschieht jeweils immer nur für den betreffenden Spieler. Für den ganzen Spielverlauf ist die Settlerklasse zuständig, die pro Spiel zwei Member der Princedomklasse besitzt. Abhängig vom Spielzug teilt diese Klasse nun die erforderlichen Aktionen auf das richtige Fürstentum auf. Dieser relativ statische Aufbau der zwei Member reicht für dieses Spiel völlig aus, da „Die Siedler von Catan-Das Kartenspiel“ nur für zwei Spieler entwickelt wurde.

6. Die Spielkarten

6.1 Klassenstruktur

Die Struktur der Spielkarten ist tief verzweigt. Für fast jede Karte im Spiel wurde eine eigene Klasse angelegt, da viele Karten sehr unterschiedliche Auswirkungen auf den Spielverlauf haben. Alle Spielkarten-Klassen befinden sich im Package „game.Cards“ und werden von der gemeinsamen Basisklasse „Card“ abgeleitet. Diese stellt Name, Kartentext, das Bild der Karte sowie eine fortlaufende ID bereit. Unterhalb der Klasse „Card“ existieren weitere Hierarchieebenen, je nach Verwendungszweck der Karte.

In der obersten Ebene werden die Karten nach „Ereigniskarten“ (CardEvent), „Aktionskarten“ (CardActivity) und „Feldkarten“ (CardField) unterteilt.

6.2 Ereigniskarten

Abbildung 12: Ereigniskarten

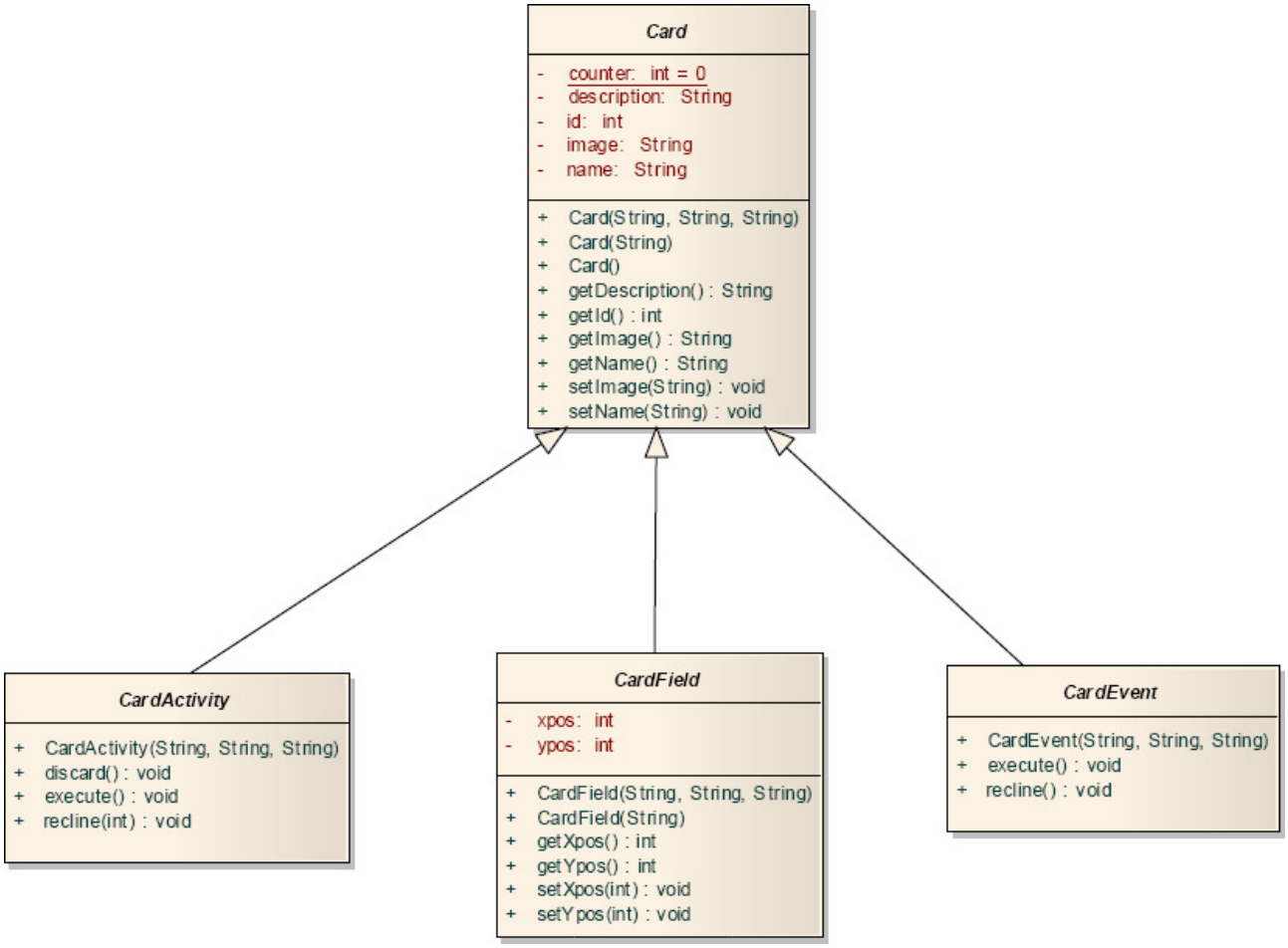


Abbildung 13: Klassenstruktur Card

Ereigniskarten liegen auf dem Zentralkartenstapel „Ereigniskarten“ und werden beim Würfelereignis „?“ gezogen, ausgeführt, und unter den Stapel zurückgelegt.

Ursprünglich war geplant, dass jede Karte eine Methode bereitzustellt, die das Ereignis der Karte ausführt. Die Klasse „CardEvent“ wurde entsprechend angelegt. Später haben wir dies allerdings verworfen, da dieses Vorgehen nicht mit der Spiellogik kompatibel war. Die Karte hätte das gesamte Spiel kennen müssen, was zu aufwändig und fehleranfällig gewesen wäre. Die Aktionen werden nun in der Spielklasse „Settler“ bzw. in der Klasse „Princedom“ ausgeführt wenn die entsprechende Karte gespielt wird.

Die Unterklassen wurden nur beibehalten um Karten leichter anlegen zu können, da Name, Beschreibungstext und Kartengrafik direkt im Konstruktor angegeben werden.

6.3 Aktionskarten

Abbildung 14: Aktionskarten

Aktionskarten liegen entweder unter einem der 5 Ausbauzentralkartenstapel oder gehören zu den Handkarten eines Spielers und können bei Bedarf ausgespielt werden. Die Aktionskarten enthalten ebenfalls jeweils eine Methode, die die entsprechende Aktion ausführen sollte. Diese wird aber wie bei den Ereigniskarten nicht genutzt, die Aktionen werden ebenfalls in der Spielverwaltungsklasse ausgeführt, wenn die entsprechende Karte gespielt wird.

6.4 Feldkarten

Abbildung 15: Fürstentum

Feldkarten sind alle Karten, die auf dem Spielfeld gebaut werden können. Diese Karten können sich ebenfalls in den Ausbaukartenstapeln liegen oder zu den Handkarten eines

Spielers gehören. Die Klasse hat Felder und Methoden für die Positionsangabe im Fürstentum. Die Feldkarten werden weiter unterteilt:

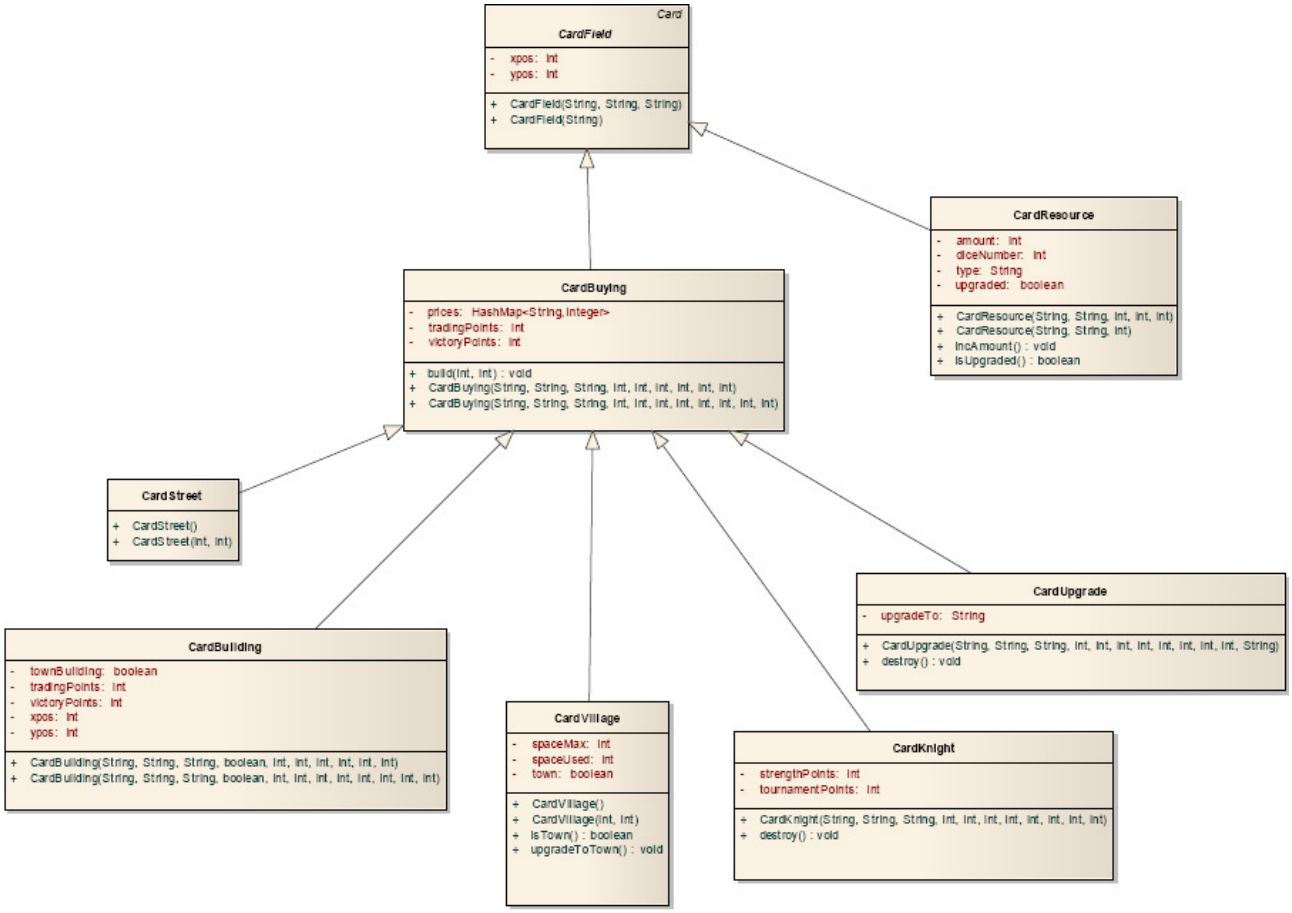


Abbildung 16: Klassenstruktur

Zunächst wird in kaufbare Karten (CardBuying) und Landschaften (CardResource) unterschieden. Alle kaufbaren Karten haben einen Preis, d.h. eine bestimmte Anzahl jedes Rohstoffs wird für den Kauf benötigt. Landschaften werden automatisch ausgelegt, wenn eine neue Siedlung gebaut wird.

Die kaufbaren Karten werden dann je nach Typ wiederum untergliedert. Es gibt Straßen (verbinden Siedlungen und Städte), Siedlungen (können zu Städten ausgebaut werden), Gebäude (haben Einfluss auf Sieg- und Handelspunkte sowie verschiedene Ereigniskarten) und Ritter (haben u.a. Einfluss auf die Rittermacht des Spielers). Weiterhin gibt es Upgrade-Karten, die Einfluss auf andere ausliegende Karten haben (z.B. verdoppelt eine Ziegelbrennerei den Ertrag benachbarter Hügellandschaften).

6.5 Verwaltung der Zentralkartenstapel

Zusätzlich gibt es die Klasse „Deck“. Diese verwaltet die Kartenstapel, die in der Mitte des Spielfeldes dargestellt werden. Je ein Objekt der Klasse verwaltet einen Zentralkartenstapel. Die Klasse „Deck“ stellt Methoden zum Ziehen der obersten Karte vom Stapel, Zurücklegen einer Karte unter einen Stapel und Mischen des Kartenstapels bereit. Außerdem existiert eine Methode, die prüft ob der Kartenstapel leer ist.

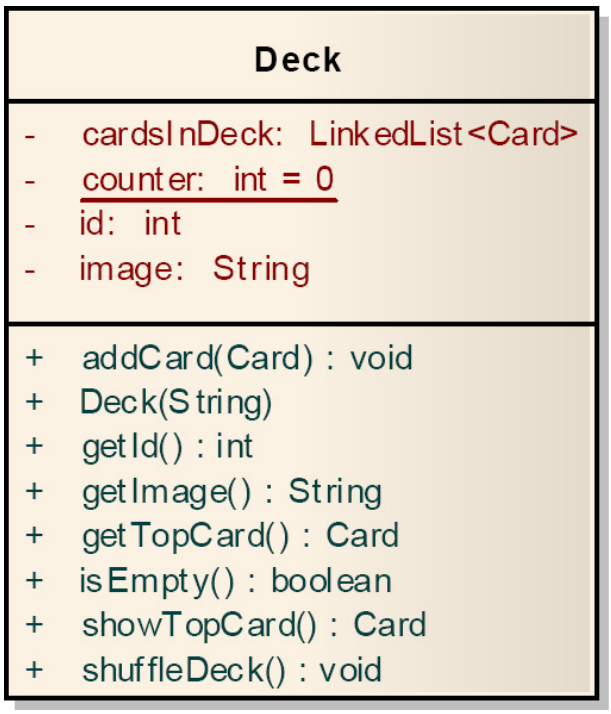


Abbildung 17: Deck

Beim Initialisieren des Spiels werden die Zentralkartenstapel regelkonform angelegt: Zunächst werden die Kartenstapel wie im Regelheft beschrieben angelegt. Dann werden die Karten auf (genauer: „unter“) den Kartenstapel gelegt (d.h. die Kartenobjekte werden ans Ende der Liste angefügt). Wurden alle Karten auf den Stapel gelegt wird der Kartenstapel noch gemischt.

Eine Besonderheit stellen die 5 Ausbaukartenstapel (ohne Symbol auf der Kartenrückseite) dar: Hier werden zunächst alle Karten auf einen temporären Kartenstapel gelegt und gemischt. Dann werden diese Karten gleichmäßig auf 5 Kartenstapel verteilt.

Aller 10 Zentralkartenstapel werden in einer HashMap „cardDecks“ in der Spielklasse verwaltet.

7. Spiellogik

Die Spiellogik wird in der Klasse „Settler“ im Package „game.Settler“ festgelegt. Die Klasse ist

abgeleitet von der allgemeinen Spielklasse „Game“. Die Spiellogik-Klasse steuert den gesamten Spielablauf, vom Initialisieren des Spiels über Ablauf der Spielzüge und Ereignisse bis hin zum Spielende. Diese Klasse ist sehr groß, obwohl alle Funktionen, die das Fürstentum eines Spielers betreffen, bereits in eine eigene Klasse „Princedom“ ausgelagert wurden, die allerdings ebenfalls sehr umfangreich geworden ist. Grund für die sehr großen Spiellogik-Klassen ist das umfangreiche und sehr detaillierte Regelwerk des Spiels.

7.1 Die Spielklasse „Settler“

Im Konstruktor der Klasse „Settler“ wird das Spiel initialisiert. Dazu gehören das Anlegen der Fürstentümer (was im Detail in der Klasse „Princedom“ geschieht), das Bereitstellen der Zentralkartenstapel sowie das Ziehen der Starthandkarten für beide Spieler.

Es werden alle Spielregeln und der Ablauf der Spielzüge überwacht und ausgeführt, u.a. das Würfeln, Bau von Siedlungen, Rittern und Gebäuden, Ausführen von Ereignis- und Aktionskarten. Dazu gehören auch Aktionen des Gegenspielers während des eigenen Spielzuges – z.B. beim Reagieren mit einer Schutz-Aktionskarte auf eine Angriffs-Aktionskarte oder bei Ereignissen wie „Ertragreiches Jahr“, die beide Spieler zugleich betreffen. Weiterhin werden am Ende eines Zuges die Anzahl der Handkarten des Spielers überprüft. Hat der Spieler weniger als die erlaubte Anzahl auf der Hand so werden neue Handkarten nachgezogen.

8. Zusammenfassung

Nach Abschluss der Programmierarbeiten lag uns ein Spiel vor, welches den von uns gesetzten Anforderungen genügte. Aufgrund einiger Schwierigkeiten, wie die Inhomogenität der zu sendenden Spieldaten oder die Zuteilung der Aufgaben zu den jeweiligen Modulen, musste oftmals innerhalb der Gruppe ein Informationsaustausch zur Problemlösung stattfinden.

Sicherlich würde nach jetzigen Gesichtspunkten die Neuerstellung des Projektes in einigen Aspekten anders betrachtet werden, so wäre es sicherlich sinnvoller dem Client auch einen Teil der Spiellogik zuzuordnen um z.B. grobe Fehleingaben abzufangen. Trotzdem stellt uns das Erscheinungsbild und die Stabilität der finalen Umsetzung zufrieden und bietet eine gute Basis für die Realisierung tiefgreifender Änderungen. Unter anderem wäre es möglich, das Spiel durch Add-ons zu erweitern. Eine weitere Ergänzung wäre auch die Portierung auf einen anderen Sendemechanismus um den Server z.B. direkt ins Internet zu stellen und auf diese Weise andere Spieler weltweit daran teilhaben zu lassen.

9. Anteile der Teammitglieder

Kapitel	Verantwortliches Teammitglied
1	Michael Körner
2	Danny Christl
3	Michael Körner
4	Danny Christl
5	Danny Christl
6	Marcus Zelend
7	Marcus Zelend
8	Michael Körner