Dokumentation

Swissdefcon-Team

Frank Müller, Oliver Wisler, Lucius Bachmann, Fabio Sulser

16. Mai 2012

Inhaltsverzeichnis

1 Grobaufbau

Unser Projekt gliedert sich in 5 grosse Pakete, alle Klassen welche Client und Server übergreifend verwendet werden, befinden sich im Paket ßhared". Der ganze Server befindet sich im Paket ßerveründ der Client im Paket "Client". Zum Ausprobieren und testen während dem Programmieren, als auch für den Unittest gibt es das Paket "test".

2 Client

2.1 Aufbau

Der Client gliedert sich in 6 Pakete, wovon 4 der Datenverwaltung dienen und 2 jeweils für die Lobby oder das Spiel zuständig sind. Eine kurze Beschreibung der Pakete findet sich in der folgenden Tabelle:

Paketname	Zweck
net	stellt Klassen für die Kommunikation und Serversuche zur Verfügung. In diesem Paket werden alle empfangenen Daten ausgewertet und mittels Event oder statischen Funktionen weitergeleitet.
events	Stellt Events zur Verfügung welche für die interne Kommunikation im Client genutzt werden. (GameEvent, ChatEvent)
data	haltet Daten welche für das Spiel oder die Lobby wichtig sind (die Spielerid, Zuordnung Spieler zu Id, etc...).
resources	Stellt eine Klasse zur Verfügung mittels derer Daten (Bilder, etc...) geladen werden können.
lobby	Beinhaltet die ganze Lobby
game	Beinhaltet das GUI für das Spiel

Tabelle 1: Unterpakete im Paket client

2.2 Datenverwaltung

Ausgehend vom Parser werden die Daten mit zwei Methoden verteilt. Daten welche nicht permanent gespeichert werden müssen (z.Bsp. Chatnachrichten) werden per Event weitergeleitet, so dass von überall her mit einem geeigneten Listener darauf zugegriffen werden kann. Daten welche langfristig gespeichert werden (zum Beispiel die Zuordnung von Spielernamen zu ihrer Id) werden von Klassen im Paket "client.data"gespeichert. Darunter fällt "PlayerManager"welcher alle Informationen bezüglich Spieler speichert und "RunningGame"welche alle Informationen zum gerade laufenden Spiel bereithält. Auf diese gespeicherten Daten kann mittels statischer Methoden jederzeit vom Spiel oder von der Lobby zugegriffen werden. frank

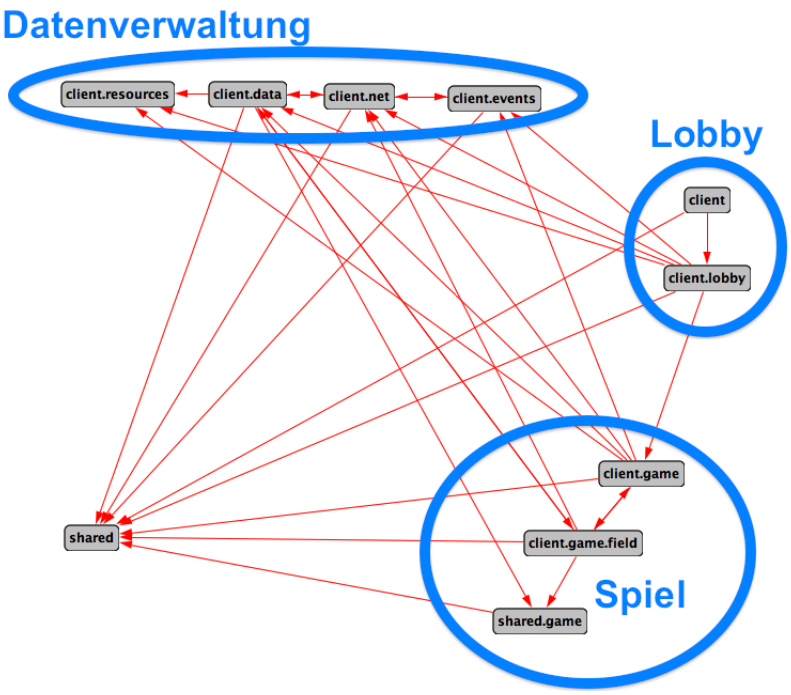

Abbildung 1: Importe zwischen den Paketen

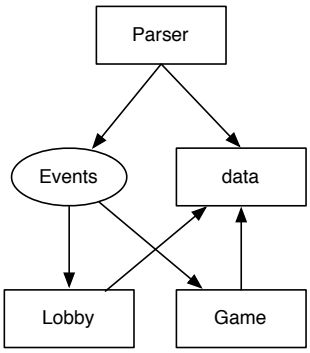

Abbildung 2: Datenverteilung

2.3 Rendering

Das Ganze Bild wird über einen doubleBuffer gezeichnet. Dazu wird aufgrund der Bildschirmgrösse die optimale Spielfeldgrösse ermittelt. Die gesetzten Objekte werden gezeichnet, fals sie bewegt wurden, wird eine Linie und eine Pfeilspitze mit Referenz auf das Objekt gezeichnet. Ein Algorithmus überprüft, ob bei einem Klick auf ein Objekt gedrückt wurde oder nicht. Falls nein, wird ein neues Objekt an der gedrükten Stelle gezeichnet, ansonsents wird die moving-range, sowie Lebenspunkte und aktuellen Wert des angewählten Objektes auf dem Spielfeld angezeigt. Die häufigkeit, in der das Spielfeld neu gezeichnet wird, hängt von der Spielsituation ab. Ist ein Objekt angewählt, so wird

das Bild alle 70 Millisekunden neu gezeichnet. Ist kein Objekt angewählt, so wird das Bild alle 200 Millisekunden neu gezeichnet. In der Animationsphase wird das Bild alle 20 Millisekunden neu erstellt. Wird die Grösse des Fenster verändert, so wird das Bild neu gerendert. Die Objekte werden als Bild gezeichnet und farbig umrandet. Jeder Spieler besitz auf der gegnerischen Karte eine andere Farbe, seine eigene ist jedoch immer Blau. Der Radar wird als dreieckige Polygone gezeichnet, die mit dem zunehmenden Winkel an Farbe verlieren. Die Objektangaben werden als Strings gezeichnet.

3 Server

3.1 Aufbau

Der Server gliedert sich in 9 Pakete, wovon 4 der Datenverwaltung dienen. Die andern Pakete enthalten Serverinterne Exceptions, das GUI, die Spiellogik und das Socket. Eine kurze Beschreibung der Pakete findet sich in der folgenden Tabelle:

Paketname	Zweck
net	Stellt Klassen für die Kommunikation und Serversuche zur Verfügung.
exceptions	Hält Serverinterne Exceptions
GamePlayObjects	Hält alle Objekte, die gebaut werden können (Bomber, Jets, Banken, usw.)
logic	Hält die Spiellogik, die die Runden bestimmt und überprüft.
parser	In diesem Paket werden alle empfangenen Daten ausgewertet und die Saten an die jeweiligen Funktionen übergeben.
players	Hält alle Spielerbezogenen Daten. (Name, ID, Socket, Server, ...)
score	Speichert eine Top-Ten der Spieler, die jederzeit aus der Lobby aufgerufen werden kann.
server	Hält alle aktiven Spiele.
UI	Hält das Benutzerinterface des Servers.

Tabelle 2: Unterpakete im Paket server

3.2 Datenverwaltung

Die vom Parser empfangenen Daten werden aufgeteilt zu Spielerdaten, Objektdaten und Serverdaten. Spielerdaten werden in der Instanz einer Player-Klasse des jeweiligen Spieler gespeichert. Objektdaten werden in der jeweiligen Instanz diese Objektes gespeichert. Serverdaten werden in einer Instanz der Server-Klasse gespeichert. Aktive Server werden vom Servermanager gehalten. Dieser entfernt auch automatisch inaktive Instanzen. 1

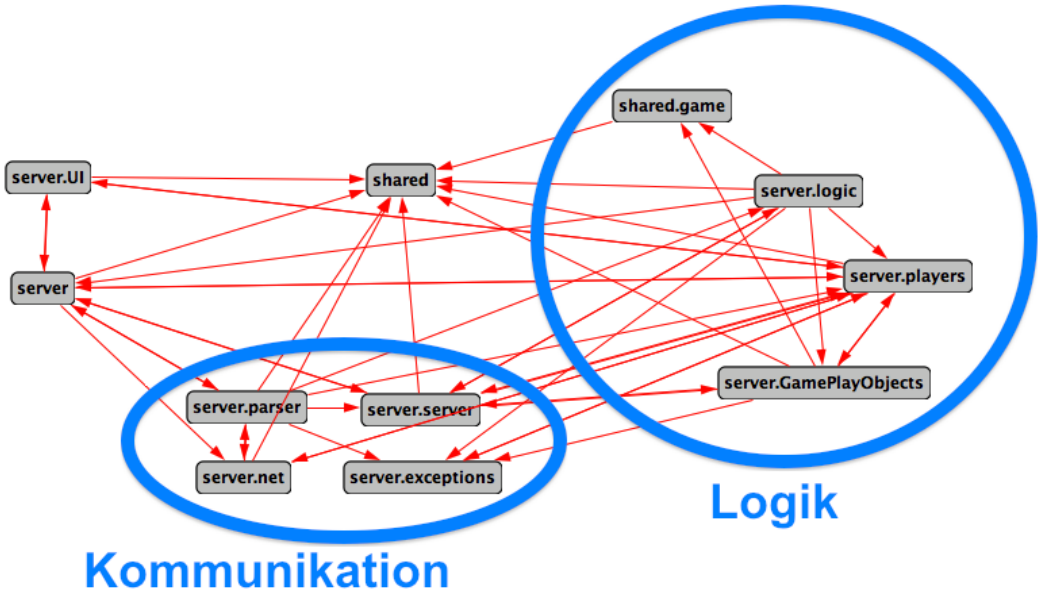

Abbildung 3: Datenverteilung

3.3 Spiellogik

3.3.1 Aufbauphase

Mit dem Start des Spiels wird der GamePlayObjectManager instanziert. Er verwaltet die GamePlayObjects. Wird ein Objekt erstellt, übergibt man dem Objekt den GamePlayObjectManager. Das Objekt, falls es an einer gültigen Position gesetzt wurde und der Spieler noch Geld hat, trägt sich dann in die Liste AllObjects und, lalalalwenn ein Defensivobjekt, in die Liste Defensives ein. Wird ein Objekt bewegt, dann wird das in die Membervariable Target eingetragen.

3.3.2 Runde

Wird jetzt die Runde gestartet, dann..

- Testet der GamePlayObjectManager zuerst, ob ein Spieler noch Population hat. Wenn nicht, werden all seine Objekte gelöscht und sein Geld auf 0 gesetzt.
- Darauf werden die Objekte geprüft, ob sie noch Lebenspunkte haben. Wenn nicht, werden sie gelöscht.
- Dann rechnet jedes Objekt aus, wohin es sich bewegen wird. Falls ein Objekt weiter als seine MovingRange bewegt werden sollte, dann bewegt es sich in Richtung Target soweit, wie es die MovingRange erlaubt. Das Ergebnis wird in moveProv gespeichert.

- Jetzt senden alle Objekte allen Objekten ihre Bewegung, und jedes Objekt testet die eingegangenen Bewegungen darauf, ob sie durch ihren Angriffsradius verlaufen. Wenn ja und angreifbar, werden sie in der Liste possibleTargets gespeichert. Dann werden die Objekte auf ihre neue Position gesetzt.
- Schiesslich führen alle Objekte ihre Angriffe aus. Die Banken geben Geld, die Reproduktionszentren Bevölkerung. Alle anderen Objekte wählen zufüllig ein Objekt aus den possibleTargets aus, schiessen darauf bis es keine Lebenspunkte mehr hat, wenn es 0 Lebenspunkte hat wählen sie das nächste Objekt, bis sie keine Munition mehr haben.

4 Kommunikation

4.1 Serverauswahl

Die Serverauswahl wird durch ein äusserst rudimentäres Discovery over Multicast bereitgestellt. Der Server sendet hier in konstantem Zeitabstand ein Multicast-Paket ins Netzwerk, das seinen Port enthält und natürlich auch seine IP (durch UDP-Packet-Header). Diese Lösung erfüllz zwar seinen Zweck, kann aber zu unnötigem Traffic innerhalb eines Netzwerkes führen. Dies zu verbessern wäre etwas für zukünftige Releases.

4.2 Verbindung

Die Verbindung basiert auf einem TCP-Socket beim Clienten und einem beim Server. Daher auf jedem Socket zwei Threads laufen (einer, der nur empfängt und einer, der nur sendet) können jederzeit daten gesendet und empfangen werden. Dies ermöglicht dem Server auch in Anwesenheit eines Routers jederzeit und instant, Nachrichten zum Clienten zu pushen. Das Socket beim Clienten sendet automatisch ein Ping, falls in den vorhergehenden 500ms keine Daten versendet wurden.

4.3 Protokoll

4.3.1 Aufbau

Jeder Protokolbefehl besteht aus 5 Buchstaben. Der erste Buchstaben bezeichnet jeweils den Bereich des Befehls, es sind dies:

- D für die Serversuche
- V für Befehle betreffend der Verbindung
- C für den Chat
- G für das Spiel

4.3.2 Befehle

Alle Protokolbefehle können im Wiki unter `http://chaos-theory.ch/CS108/wiki/doku.php?id=protocol` abgerufen werden.

5 Andere Pakete

5.1 shared

Hier sind diverse Klassen, welche sowohl vom Server als auch vom Client benutzt werden. Darunter fällt das Protokoll, ein InputValidator, welcher Eingaben validieren kann, Eine Klasse welche Log Funktionen zur Verfügung stellt, diverse Einstellungen (Spiel und Kommunikation).

6 Unittest

6.1 Die Klasse

Funktionen der Klasse Bank:

- Der Konstruktor:

  Wenn der Spieler zuwenig Geld hat-> throw GameObjectBuildException

  Wenn der Spieler das Objekt nicht in seinem Feld setzt-> throw GameObject-BuildException

  Wenn der Spieler genug Geld hat und das Objekt in seinem Feld setzen will-> Füge dich in die Liste aller GamePlayObjects vom Server ein.

- die Methode damage(int damPoints)

  Reduziere die Healthpoints um damPoints

- Die Methode attack()

  Füge dem Besitzer das soviel Geld hinzu, wie in den Settings definiert.

6.2 Der Test

Getestet wurde:

- Ob es eine Exception gibt, wenn man eine Bank ohne Geld an ausserhalb des eigenen Feldes setzt. Wenn ja, ok, Wenn nein, fail.

- Ob es eine Exception gibt, wenn man eine Bank ohne Geld an innerhalb des eigenen Feldes setzt. Wenn ja, ok, Wenn nein, fail.

- Ob es eine Exception gibt, wenn man eine Bank ohne Geld an ausserhalb des eigenen Feldes setzt. Wenn ja, ok, Wenn nein, fail.
- Ob es eine Exception gibt, wenn man eine Bank mit Geld an ausserhalb des eigenen Feldes setzt. Wenn ja, ok, Wenn nein, fail.
- Ob es eine Exception gibt, wenn man eine Bank mit Geld an innerhalb des eigenen Feldes setzt.Wenn ja, fail, Wenn nein, ok.
- Wenn die Bank korrekt erstellt wurde, ist sie dann in der Liste der Objekte: Wenn ja, ok, Wenn nein fail.
- Wenn die Bank mit 100 Schadenspunkten beschädigt wurde, hat sie nacher 100 Healthpoints weniger. Wenn ja, ok, Wenn nein fail.
- Wenn die attack Methode aufgerufen wurde, hat der Spieler soviel Geld mehr, wie in den Settings definiert? Wenn ja, ok, Wenn nein fail.