Dokumentation NetBomb

Stefan Künzle <stefan.kuenzle@hsr.ch>
René Herrmann <rene.hermann@hsr.ch>
Michael Egli <michael.egli@hsr.ch>
Urs Heimann <urs.heimann@hsr.ch>

Betreuer: Daniel Keller

Erstellt am 12. Juli 2002

Typesetting by LaTeX

Inhaltsverzeichnis

Teil I

Iteration 1

Kapitel 1

Vision

1.1 Ausgangslage und Motivation

Bomberman, ein unterhaltsames Spiel für mehrere Spieler soll netztauglich und auf Linux portiert werden. Bei herkömlichen Windows-Versionen für mehrere Spieler gestaltet sich die gleichzeitig an einer Tastatur erfolgende Bedienung als äusserst unkonfortabel. Ebenso stehen oft nicht gleich drei weitere Mitspieler vor Ort zur Verfügung, darum soll eine Netzwerk taugliche Version geschaffen werden. Das Spiel hat keine komplizierten Spielregeln. Es soll mit einer ganzheitlich einfachen Handhabung realisiert werden, damit man unverzüglich in den Mehrspieler-Spielgenuss eintauchen kann.
Da wir unser eigener Auftraggeber sind, begründet sich unsere Motivation auch in der Bewältigung technologischer, software-engineering orientierter und zwischenmenschlicher Aufgaben. Herausforderungen, die wir uns selber aufstellen.

1.2 Features

1.2.1 Iteration 1

- 1 Spielermodus
- Spielfeld mit Wänden und Mauern
- Spielfigur kann man bewegen
- optional: Die Bewegungen des Spielers werden übers Netzwerk übertragen und auf einem anderen Rechner angezeigt

1.2.2 Iteration 2

- Es können Bomben gelegt werden, die explodieren
- Spieler können sterben
- Es können 4 Spieler zusammen übers Netzwerk spielen

1.2.3 Optionen der Iteration 2

- Es gibt Icons die der Spielfigur spezielle Fähigkeiten verleihen
- Soundeffekte sind zu hören

- Hintergrundmusik ist zu hören
- es gibt einen Pausemodus
- eine Highscore der besten Spieler ist verfügbar

1.3 Herausforderungen

1.3.1 Implementation

- Animierte Grafik und Sound auf Linux
- Netzwerk Implementation
- Ablauf der Synchronisation des Spieles zwischen verschiedenen Rechnern

1.3.2 Technologie

- Linux
- Dokumentation mit LaTeX
- Design Software Together
- Qt (Graphikbibliothek für KDE Desktop)

1.3.3 anderer Art

- **Zielgerichtetes Arbeiten**
  Durch die begrenzte, pro Woche zur Verfügung stehende Zeit(Ziel max. 4-8) und die vielseitige Aufgabenstellung, müssen wir zwangsläufig streng zielgerichtet arbeiten, um die Meilensteine erfüllen zu können.
- **Teamwork (2≪Teamitglieder)**
  Ein Team mit mehr als zwei Personen erlaubt keine *philosophischen* Gruppendiskussionen mehr. Es kann weder auf eine Traktandenliste für Sitzungen, noch auf eine streng sachliche pro/contra Argumentation während einer Diskussion verzichtet werden.
- **Kompromissbereitschaft**
  Es muss von Perfektionismus zu notwendiger Zweckmässigkeit hingearbeitet werden.
- **Eigeninitiative**
  Unser Team hat praktisch keine Erfahrungen aus grösseren Projekten mit grösseren Teams. Gefragt sind Eigeninitiative zur Arbeit an eigener Teambereitschaft und Lernbereitschaft (nicht nur für Technologie sondern auch im Teambildungsprozess und Selbstorganisation).
  Anpacken ist angesagt, aber nicht blind ohne Priorisierung und Ziel sondern mit Blick zum Horziont und vereinten Kräften.

chapterAnforderungsspezifikation

1.4 Einführung

1.4.1 Zweck

Dieses Kapitel legt die Anforderungen an das Programm NetBomb fest.

1.4.2 Gültigkeitsbereich

Semesterarbeit Software Engineering, Sommersemester 2002

1.4.3 Definitionen, Akronyme, Abkürzungen

Siehe Anhang A auf Seite 88.

1.5 Allgemeine Beschreibung

1.5.1 Spielregeln

Ziel des Spiels ist es, die gegnerische Spielfigur mittels einer Bombe und dessen Bombenstrahl zu eliminieren. Im Spiel gibt es zwei Spielfiguren, eine fixe Anzahl Mauern und eine variable Anzahl Wände. Die Spielfigur kann Wände sprengen, Mauern sind unzerstörbar. Die Spielfigur kann weder durch Wände noch durch Mauern gehen. Unter gesprengten Wänden können sich Bomben oder Feuersymbole befinden. Das Aufheben einer Bombe erlaubt der Spielfigur das Legen einer zusätzlichen Bombe in Serie. Das Aufheben einer Flamme erlaubt der Spielfigur einen um ein Feld längerer Bombenstrahl. Die zwei Spielfiguren sind gegenseitig transparent. Sie können sich kreuzen.

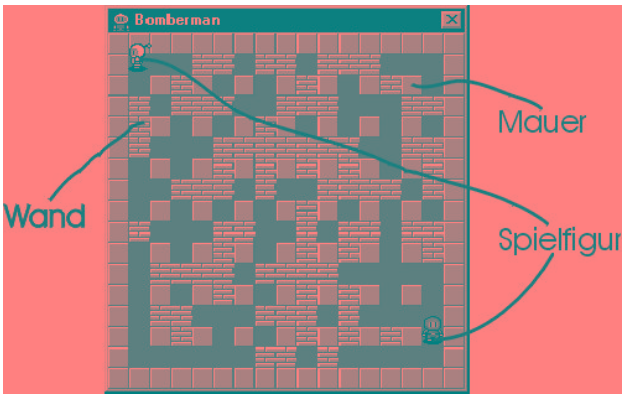

Abbildung 1.1: Bomberman Spiel

1.5.2 Benutzergruppen

Alle Menschen ob jung oder alt, weiblich oder männlich, die Freude an Computerspielen haben.

1.5.3 Mögliche Erweiterungen

- Sound Unterstützung
- spezielle Powerup-Icons, die von der Spielfigur aufgenommen werden können.
- Highscore Anzeige

1.5.4 Zu erwartende Probleme

wurden bereits im Dokument Projektplan eingetragen.

1.5.5 Annahmen

Da die von uns verwendeten Technologien für uns neu sind, sind wir uns bewusst, dass das Risiko vorhanden ist, dass wir die Anforderungen nicht vollständig erfüllen können. Dieses Dokument wurde in der Annahme geschrieben, dass wir die auftretenden Probleme lösen können. Ansonsten werden wir die Anforderungen zusammen mit dem Betreuer anpassen.

1.5.6 Abhängikeiten

Funktionalität Qt-Bibliothek und KDE-Bibliothek.

1.6 Anforderungen im Einzelnen

1.6.1 Funktionale Anforderungen Iteration 1

Spiel

Referenz	Funktion	Priorität
A1.1	Spiel starten	1
A1.2	Spiel beenden	1
A1.3	Spielregeln überwachen	1

Tabelle 1.1: Spiel Funktionen Iteration 1

Spielfigur

Referenz	Funktion	Priorität
A2.1	Figur bewegen	1

Tabelle 1.2: Spielfigur Funktionen Iteration 1

Spielfeld

Referenz	Funktion	Priorität
A3.1	Hintergrund zeichnen	1
A3.2	Mauern und Wände zeichnen	1
A3.3	Spielfigur zeichnen	1

Tabelle 1.3: Spielfeld Funktionen Iteration 1

1.6.2 Funktionale Anforderungen Ieration 2

Spiel

Referenz	Funktion	Priorität
A1.4	Spielstand aktualisieren	2
A1.5	Highscore speichern	2

Tabelle 1.4: Funktionen Spiel Iteration 2

Spielfigur

Referenz	Funktion	Priorität
A2.2	Bombe legen	2
A2.3	sterben	2

Tabelle 1.5: Funktionen Spielfigur Iteration 2

Spielfeld

Referenz	Funktion	Priorität
A3.4	Wand entfernen	2

Tabelle 1.6: Funktionen Spielfeld Iteration 2

Bombe

Referenz	Funktion	Priorität
A4.1	explodieren	2
A4.2	Reichweite berechnen	2

Tabelle 1.7: Funktionen Bombe Iteration 2

Netzwerk

Referenz	Funktion	Priorität
A5.1	Server starten	2
A5.2	Client anmelden	2
A5.3	Spielelement Position übermitteln	2
A5.4	Spielsituation synchronisieren	2

Tabelle 1.8: Funktionen Netzwerk Iteration 2

1.6.3 Use Cases Iteration 1

UC01 Spiel starten

Auslösender Aktor	Spieler
Zweck / Ziel	Spielfeld und Spielelemente zeichnen, Netzwerkverbindung aufbauen
Priorität	1
Style	fully dressed
Anforderungen	A1.1, A1.2, A3.1, A3.2, A3.3
Vorbedingung	-
Nachbedingung	Spielfeld und Spielelemente gezeichnet, Netzwerkverbindung aufgebaut.
Bemerkungen	-

Tabelle 1.9: UC01 Spiel starten

Grundlegender Ablauf

Aktor	**System**
1. Benutzer startet neues Spiel	
	2. Leveldaten einlesen
	3. Spielfeld zeichnen
	4. Spielelemente zeichnen
	5. wartet auf Benutzereingabe

Erweiterungen
∗a zu jeder Zeit kann der Spieler das Spiel beenden

UC02 Spielfigur bewegen

Auslösender Aktor	Spieler
Zweck / Ziel	Aktor kann Spielfigur in horizontaler oder vertikaler Richtung bewegen
Priorität	1
Style	fully dressed
Zu erfüllende Anforderungen	A1.3, A2.1, A3.3
Vorbedingungen	UC01
Nachbedingungen	Die Spielfigur wurde um ein Feld verschoben.
Bemerkungen	Dieser UC kann 1 oder n mal ausgeführt werden.

Tabelle 1.10: UC02 Spielfigur bewegen

Grundlegender Ablauf

Aktor	**System**
1. Der Aktor verschiebt die Spielfigur um ein Feld nach links, rechts, oben oder unten.	
	2. zeichnet die Figur auf dem neuen Feld, sofern das Zielfeld nicht einer Wand oder einer Mauer enstpricht.

1.6.4 Optional

Spiel

Referenz	Funktion	Priorität
A1.6	Spiel pausieren	3
A1.7	Soundeffekte abspielen	3
A1.8	Musik abspielen	3

Tabelle 1.11: optionale Funktionen Spiel

Spielfigur

Referenz	Funktion	Priorität
A2.4	Bombe-Powerup aufnehmen	3
A2.5	Flamme-Powerup aufnehmen	3

Tabelle 1.12: optionale Funktionen Spielfigur

Spieloptionen

Referenz	Funktion	Priorität
A6.1	Sound ein/aus	3
A6.2	Highscores anzeigen	3
A6.3	Spielername eingeben	3

Tabelle 1.13: Spieloptionen

1.6.5 Leistungs- und Mengenanforderungen

Leistungsanforderungen

Um die Spielbarkeit übers Netzwerk zu gewährleisten, muss der Spielstatus von allen Spielern mindestens alle 150ms synchronisiert werden.

Mengenanforderungen

Keine.

1.6.6 Anforderungen an Schnittstellen

Benutzerschnittstelle

Das System ist mit dem Keyboard und der Maus bedienbar.

Software Schnittstellen

Qt, KDE-Library

1.6.7 Randbedingungen für den Entwurf

Übereinstimmungen mit Normen

SE01/02

Einschränkungen bezüglich Software

Lauffähig unter KDE 2.2

Einschränkungen bezüglich Hardware

Lauffähig unter allen UNIX-Derivaten, die KDE unterstützen. Die Leistungsanforderungen gelten für ein Netzwerk (mind. 10Mb/s) ohne zusätzlichen Datenverkehr.

1.6.8 Merkmale

Benutzbarkeit

Die Bedienung des Programms entspricht den gängigen KDE-Programmen:
http://developer.kde.org/documentation/standards/kde/style/basics/index.html

1.6.9 Andere Anforderungen

Inbetriebnahme / Installation

Standardinstallationsweg eines Linux-Quellcodes (configure, make, make install)

Konfigurierbarkeit

Alle Konfigurationen werden gespeichert. Die IP-Adresse des Spielservers kann eingegeben werden. Optional kann der Spielername eingegeben werden.

Kapitel 2

Supplementary Specification

2.1 Functionality

2.1.1 Error handling

Fehler werden dem Spieler mittels einer Fehlermeldung gemeldet. Die Fehler werden nicht persistent in einer log-datei gespeichert.

2.1.2 Security

Das Programm hat keine speziellen Sicherheitsvorkehrungen. Es kommuniziert über ein IP-Netz mit den Programmen der anderen Spieler.

2.2 Performance

Mit den unter 1.6.5 auf Seite 14 angegebenen Leistungsanforderungen soll ein angenehmes Spielverhalten gewährleistet werden.

2.3 Supportability

2.4 Free open source components

Als Linux-Software-Developers liegt es Nahe, OpenSource Technologien zu verwenden. (siehe dazu Projektplan, Kapitel Technologien)

2.5 Developer Guidelines

2.5.1 Code Guidelines

- Methodennamen beginnen mit Kleinbuchstaben
- Klassennamen beginnen mit Grossbuchstaben
- Variabeln beginnen mit Kleinbuchstaben
- Vor und nach Operations-Zeichen wird ein Abstand gemacht.
- Nach zusammengehörigen Code-Blöcken eine leere Zeile einfügen.

- Bei Klassen die öffnende geschweifte Klammer rechts neben dem Klassennamen und die schliessende auf eine eigene Zeile. Für alle anderen Fälle die öffnende und schliessende geschweifte Klammer auf eine eigene Zeile.
- Code innerhalb geschweifter Klammern wird eingerückt.
- Falls Tabulatoren verwendet werden, in der Entwicklungsumgebung definieren, dass dafür Leerzeichen eingefügt werden.
- Default-Einrückung: Zwei Leerzeichen
- jede Kommentarzeile mit zwei slashes (//) beginnen.

2.5.2 GUI Guidelines

Folgt später in einem speraten Dokument.

Kapitel 3

Domainanalyse

3.1 Konzeptionelles Modell

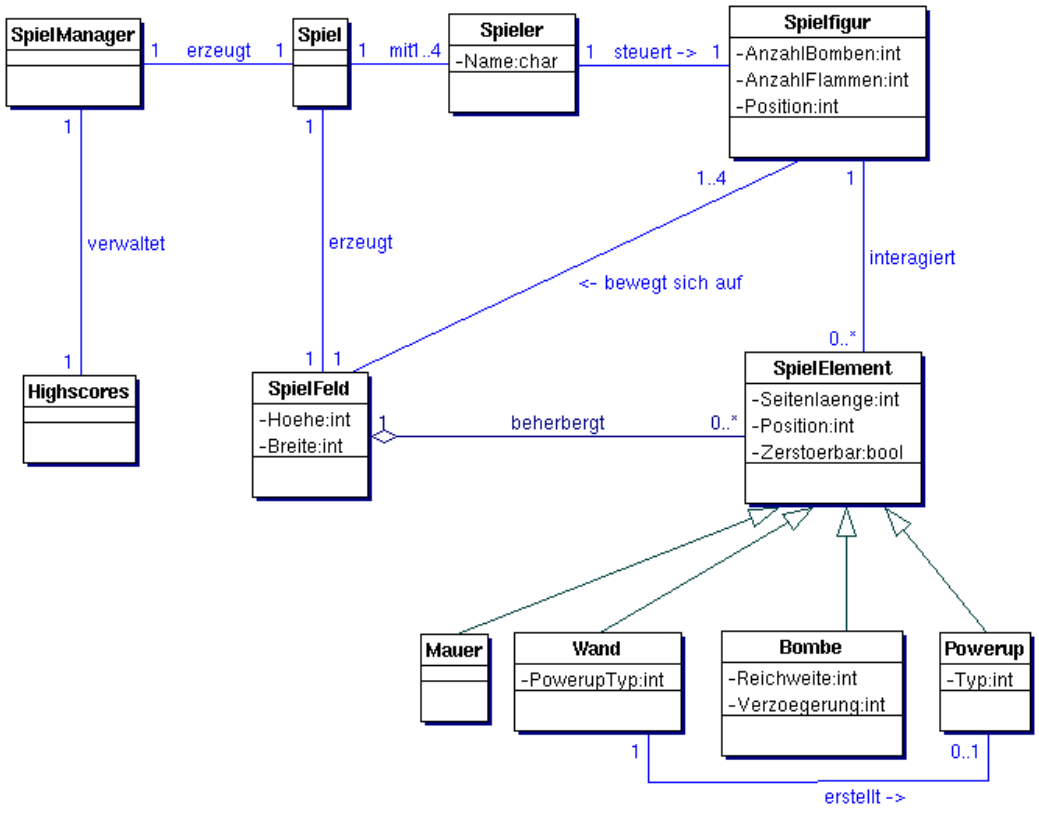

Abbildung 3.1: Domainmodell

3.1.1 Spielmanager

Der SpielManager ist für die Initialisierung des Spiels zuständig. Er kennt die zwei Spieler und weitere für das Spiel notwendige Parameter.

3.1.2 Spiel

Das Spiel erzeugt das Spielfeld und positioniert die Spielelemente für den Start des Spiels.

3.1.3 Spielfeld

Das Spielfeld ist der Hintergrund auf dem das eigentliche Spiel stattfindet. Es hat eine bestimmte Höhe und Breite die zu beginn festgelegt werden. Auf dem Spielfeld befinden sich die Objekte des Spielfeldes:Spielelemente und Spielfigur. Alle Objekte des Spielfeldes haben eine Position. Das Spielfeld kennt alle Objekte und ihre Positionen.

3.1.4 Spielelement

Das Spielelement dient als Oberklasse für sämtliche sich auf dem Spielfeld befindlichen Objekte, ausser der Spielfigur. Die einheitliche Schnittstelle erleichtert die Realisierung einiger Funktionen wie die Zerstörung von Objekten usw. Ein Objekt des Spielfeldes belegt normalerweise ein Feld auf dem Spielfeld. Es kann über das Spielfeld die Belegung der benachbarten Felder abfragen.

3.1.5 Spielfigur

Die Spielfigur ist das einzige Objekt des Spielfeldes das sich auf dem Spielfeld bewegen kann. Sie kann mit den Spielelementen kommunizieren. Die Spielfigur kann zum Beispiel Bomben erzeugen (legen) oder Powerups aufnehmen. Andererseits wird sie von Mauern und Wänden am weitergehen gehindert.

3.1.6 Mauer

Die Mauer wird vor Spielbeginn erzeugt und bleibt während dem ganzen Spiel an ihrem Platz. Sie kann nicht zerstört werden, trotzt also auch explodierenden Bomben.

3.1.7 Wand

Die Wand wird ebenfalls zu beginn erzeugt, kann aber durch Bomben gesprengt werden und verschwindet in diesem Fall vom Spielfeld. Eine Wand kann (muss aber nicht) ein Powerup beherbergen. Dieses bleibt auf dem Feld liegen falls die Wand zerstört wird.

3.1.8 Bombe

Die Bombe wird von der Spielfigur erzeugt. Dabei werden die Attribute wie Reichweite und Verzögerungszeit gesetzt. Danach ist die Bombe ein eigenständiges Spielelement und explodiert entweder nach Ablauf der Verzögerungszeit oder wenn sie von einer anderen Bombe gesprengt wird. Nach dem Explodieren meldet sie ihr Ableben der Spielfigur, die sie erzeugt hat.

3.1.9 Powerup

Ein Powerup ist entweder vom Typ Bombe (eine Bombe mehr in serie) oder vom Typ Flamme (grössere Reichweit) und ist zu Beginn unter einer Wand verborgen. Es tritt erst in erscheinung wenn die Wand durch eine Bombe zerstört wurde. Betritt eine Spielfigur dasselbe Feld, nimmt sie das Powerup auf und es wird gelöscht. Wird das Powerup von einer explodierenden Bombe erfasst, wird es zerstört.

3.1.10 Spieler

Der Spieler hat einen Namen und steuert die Spielfigur. Er kann sie in alle 4 Richtungen bewegen und Bomben legen.

Kapitel 4

Software Architektur

4.1 Logische Sicht

Das System besteht aus primär 4 Schichten (3 Schichten - Architektur plus Netzwerk). Diese beinhalten:

Schicht 1 Graphisches User Interface (GUI)

Schicht 2 Problem Domain (PD)

Schicht 3 Datenhaltung (DH)

Schicht 4 Netzwerk (bestehend aus Client und Server)

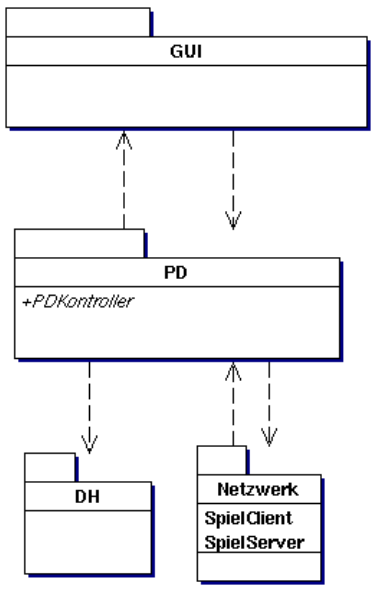

Abbildung 4.1: Schichtenmodell

Diese Architektur haben wir gewählt um eine klare Trennung zwischen den einzelnen Programmteilen zu erreichen.

Das graphische User Interface ist die Schnittstelle zum Benutzer. Über dieses kann er Eingaben machen oder in unserem Fall erfolgt die Steuerung der Spielfigur über das GUI.
Die PD ist die Schicht zwischen GUI und DH. Das heisst sie bekommt und verarbeitet Befehle vom GUI, schreibt Daten in die DH und ruft Funktionen in der Netzwerkschicht auf.
Die Datenhaltung ist dafür zuständig, Daten, die konsistent sein müssen zu speichern, damit sie bei einem Neustarten des Spiels wieder zur Verfügung stehen.
Die Netzwerkschicht regelt die Datenübertragung zwischen dem Server und den Clients.

4.2 Netzwerk

Das Netzwerk besteht aus einem Server und bis zu vier Clients die miteinander kommunizieren. Dabei kann jeder Client auch Server sein, das heisst zu Beginn des Spiels entscheidet der Spieler, ob er Server und Client oder nur Client ist. Es kann nur ein Spieler Server sein. Ist ein Spieler Server und Client, werden die Daten der Mitspieler zu ihm übermittelt. Das geschieht folgendermassen:
Die aktuellen Bewegungen eines Clients, also eines Spielers werden zum Server übermittelt. Diese Daten werden vom Server entgegengenommen dieser berechnet damit die aktuelle Position und allfällige Aktionen des Clients. Diese berechnete Postion schickt der Server dann *allen* Clients zurück. Das heisst, jeder Client bekommt vom Server einen Snapshot. Zusätzlich zur Position beinhaltet dieser wichtige Daten wie zum Beispiel ob eine Bombe gelegt worden ist oder ein Powerup aufgenommen wurde. Mit diesen Angaben berechnet der Client selbständig, was auf dem Spielfeld passiert. Er macht also die selben Berechnungen wie der Server. Falls ein Spielelement gelöscht werden muss, versucht der Client das zu machen. Da er aber nicht selbständig Elemente löschen darf, wartet er auf den Befehl des Servers, das entsprechende Element zu löschen. Der Client führt also nur eine Art Dummy-Funktion aus. Damit erreichen wir eine bessere Performace wie mit dem Prinzip, bei dem alle Daten vom Server zum Client geschickt werden und zudem können wir damit sicherstellen, dass alle Clients den selben Spielstand haben. Die Verbindung läuft über TCP, was das Ankommen der Pakete sicherstellt.
Der Server sowie der Client wurde mit dem Reactor Pattern implementiert. Dieses wird nachfolgend noch genauer erklärt.

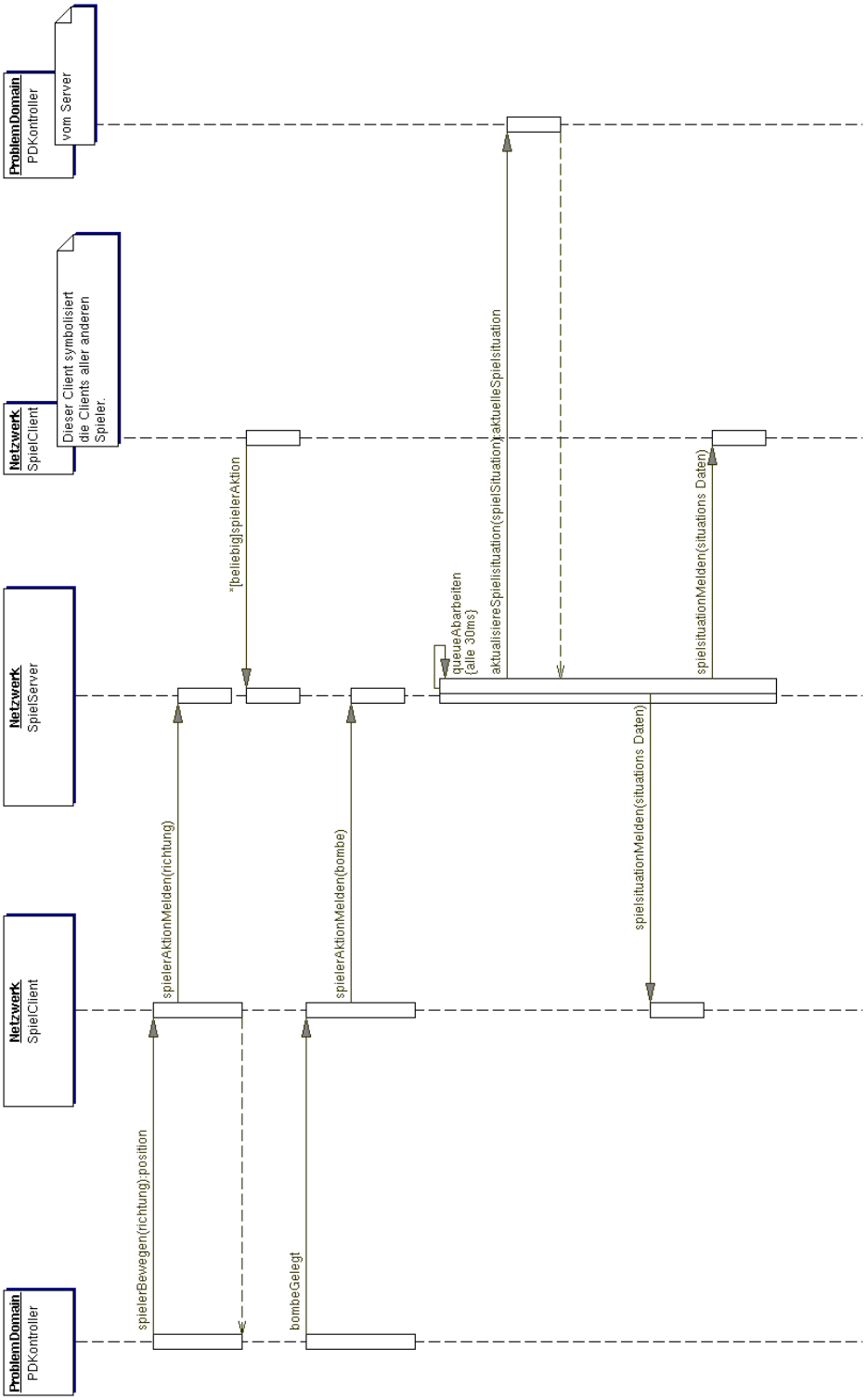

Abbildung 4.2: Interaktionsmodell Netzwerk

Kapitel 5

Designmodell

5.1 GUI Externes Design

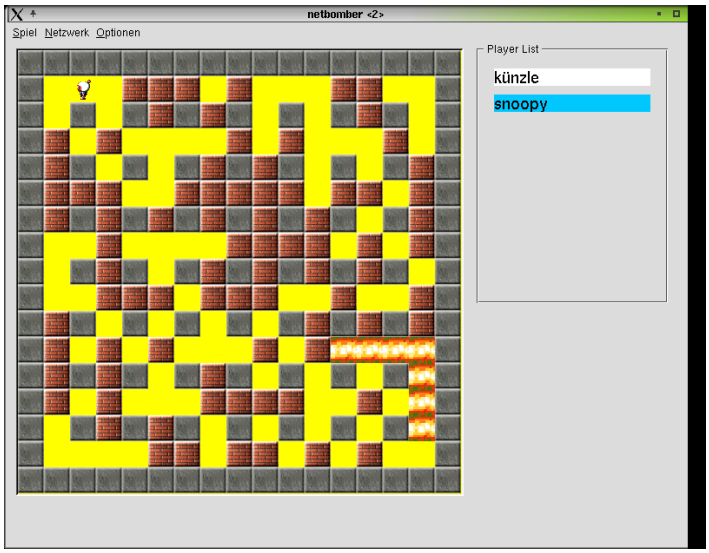

Abbildung 5.1: Snapshot des GUIs

Wie aus dem Snapshot ersichtlich ist, ist das externe GUI Design für die Iteration 1 stark

vereinfacht. Uns war wichtig, dass die grundlegenden Funktionalitäten (Spieler bewegen, collision detection) einwandfrei funktionieren. Der Spieler kann über das Menu Spiel ein neues Spiel starten oder die ganze Applikation beenden. Mit den Pfeiltasten auf, ab, links und rechts steuert er die Spielfigur.

5.2 GUI Klassendiagramm

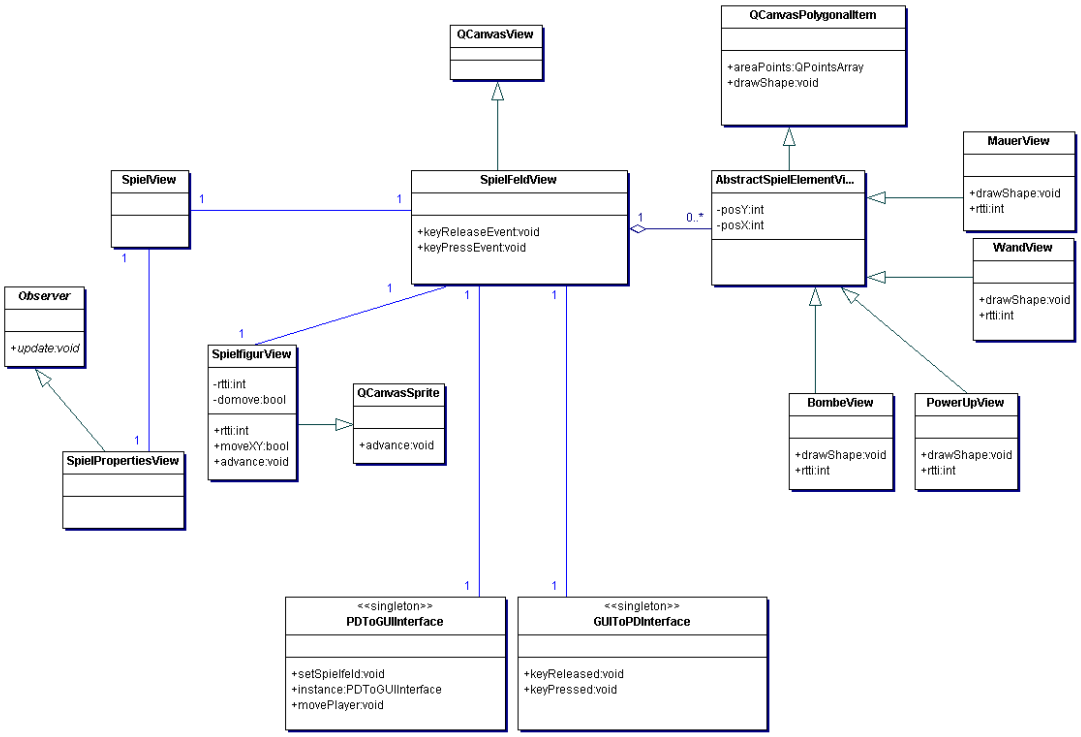

Abbildung 5.2: Klassendiagramm GUI

5.3 GUI Sequenzdiagramme

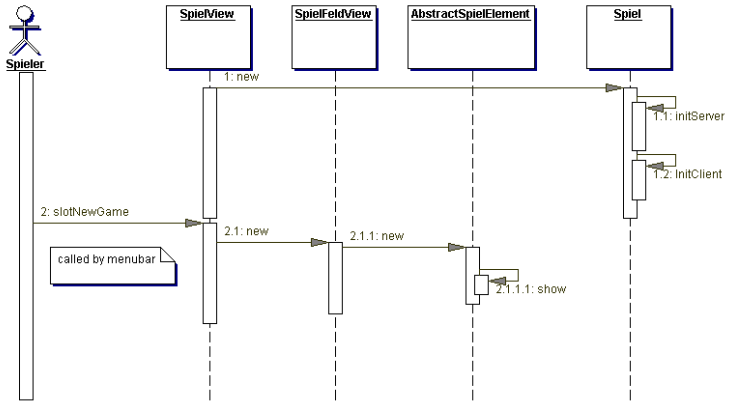

Abbildung 5.3: Sequenzdiagramm für neues Spiel

5.4 GUI Klassenbeschreibung

5.4.1 Klasse SpielView

Die Klasse SpielView enthält einzig allein ein Menu, welches zum Starten eines neuen Spiels, zum Beenden des Spiels, oder um das Optionsfenster zu öffnen, dient. Beim ersten Fall wird eine neue Instanz der Klasse SpielFeldView erzeugt.

5.4.2 Klasse SpielFeldView

Sie ist sozusagen die Kernklasse im GUI Design. Diese Klasse verwaltet alle sich auf dem Spielfeld befindenen Elementen (Aggregation zu AbstractSpielElementView). Um das zu erreichen, wird sie von der Klasse QCanvasView abgeleitet. QCanvasView ist die Präsentationsklasse für QCanvasItems, also für einzelne Grafikelementen. Auch das Keyboard-Eventhandling erfolgt in dieser Klasse. Falls Keyboard-Events auftreten, werden diese abgefangen, und nur die für das Spiel relevanten Steuersignale werden der Klasse PDToGUIInterface weitergeleitet, wo sie dann von der PD verarbeitet werden. Gleichzeitig erhält sie von der Klasse GUIToPDInterface Methodenaufrufe, welche das Verändern des UI's zur Folge haben (Verschieben der Spielfigur, entfernen / hinzufügen von Mauern etc.). Man kann sagen, die Klasse SpielFeldView ist völlig intelligenzlos, sie nimmt nur Befehle von der Interfaceklasse entgegen, oder leitet an diese Steuersignale weiter.

5.4.3 Klasse SpielfigurView

Diese Klasse repräsentiert die Spielfigur. Sie wird im Konstruktor der Klasse SpielFeldView erzeugt. Ihre Koordinaten erhält sie ebenfalls von der Klasse SpielFeldView.

5.4.4 Klasse AbstractSpielElementView

Diese Klasse wurde von der Klasse QCanvasPolygonalItem abgeleitet. Sie ergänzt diese Klasse nur mit den Koordinaten. Gleichzeitig dient sie als Oberklasse für alle auf dem Spielfeld befindenen

Elementen, ausser der Spielfigur. Dies ist notwendig, um die Aggregation zwischen SpielFeldView und AbstractSpielElement zu erreichen.

5.4.5 Klasse MauerView

Sie räpresentiert auf dem Spielfeld eine unzerstörbare Mauer. Das Attribut rtti steht für run time type information. Sie dient zur Identifikation.

5.5 PD Klassendiagramm

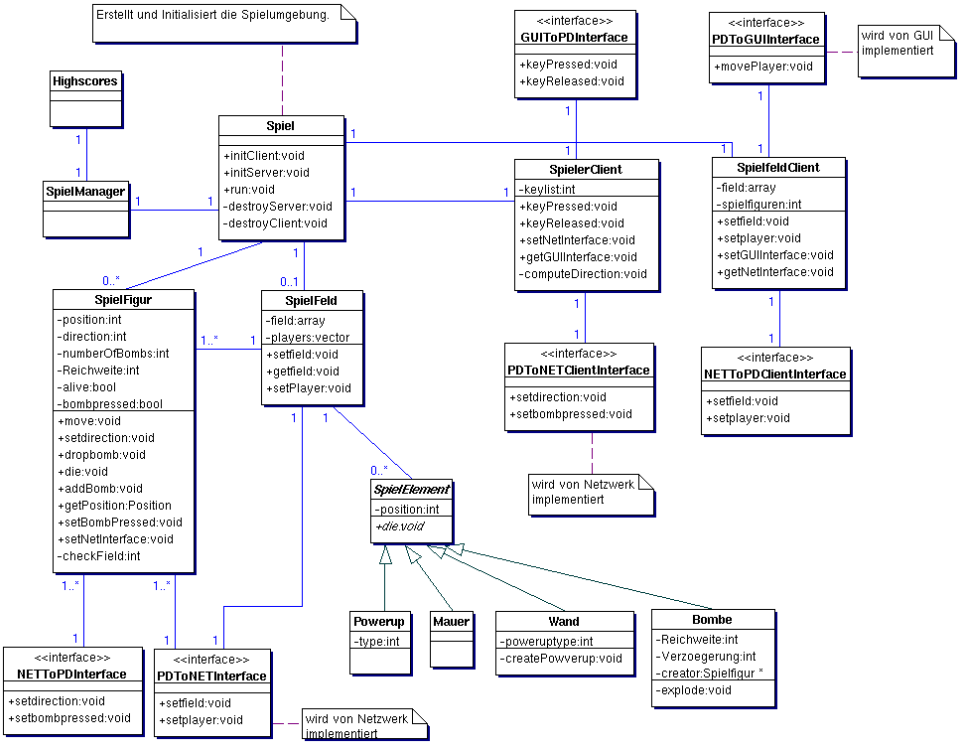

Abbildung 5.4: Klassendiagramm PD

5.6 PD Sequenzdiagramme

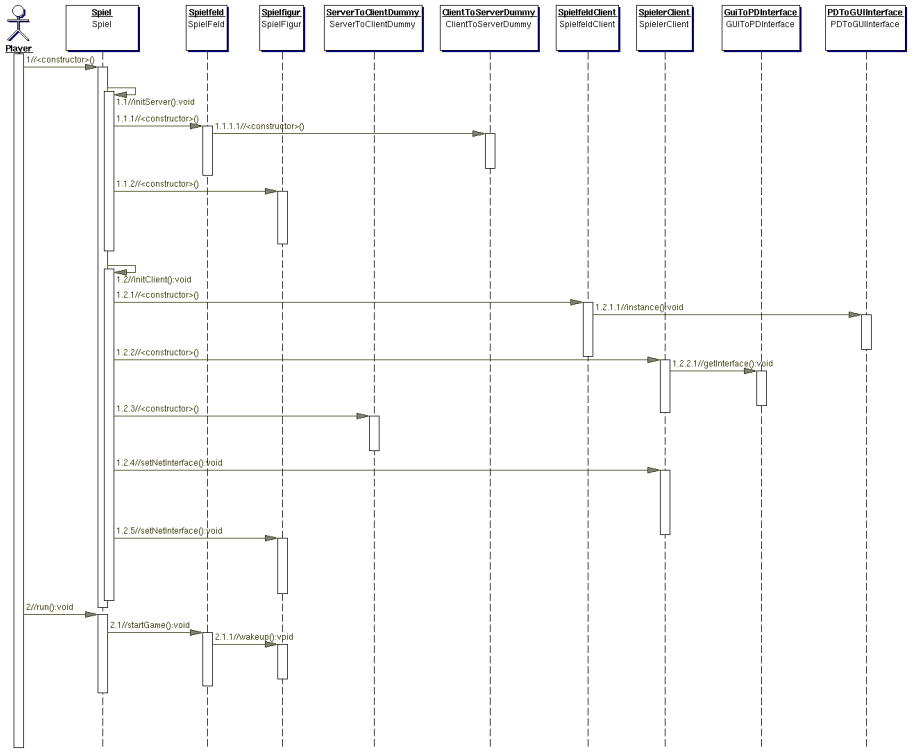

Abbildung 5.5: Sequenzdiagramm für neues Spiel

5.7 PD Klassenbeschreibung

Die endgültige Version des Spiels soll über ein Netzwerk gespielt werden. Deshalb wird die PD in Server und Client unterteilt. Die Klassen des Servers sind für die Berechnung des gesammten Spielverlaufs (Positionen, Treffererkennung, ...) verantwortlich. Der Client konvertiert die Steuerbefehle und schickt sie übers Netzwerk an den Server. Der Server berechnet die Auswirkungen und schickt die Änderungen zurück an alle Clients. Der Client reicht die Änderungen wiederum an das User Interface weiter. Jeder Spieler hat auf seinem Rechner eine Instanz des Clients, aber nur einer hat zusätzlich noch eine Instanz des Servers.

5.7.1 Klasse SpielManager

Der Spielmanager sucht vor dem Spiel andere Spieler, die sich bei ihm anmelden können. Er startet (erzeugt) anschliessend das Spiel mit den angemeldeten Spielern.

5.7.2 Klasse Spiel

Die Klasse Spiel erzeugt die restliche Spielstruktur (Spielfeld, Spielfiguren) je nach Anzahl Spieler. Es wird immer ein Client erstellt, und beim Spielführer zusätzlich noch ein Server. Sie ist dafür verantwortlich, dass das Spiel mit allen Clients synchronisiert ist bevor das Spiel gestartet wird.

Funktionen

+initClient() : void	Initialisiert die Client-Umgebung.
+initServer() : void	Initialisiert die Server-Umgebung.
+run() : void	Synchronisiert und startet das Spiel.
-destroyClient() : void	Räumt die Client-Umgebung nach Spielende auf.
-destroyServer() : void	Räumt die Server-Umgebung nach Spielende auf.

5.7.3 Klasse GUIToPDInterface (Singleton)

Über diese Schnittstelle sendet das User Interface die Tastatureingaben des Spielers an die PD.

Funktionen

+keyPressed(int key) : void	Taste key wurde gedrückt.
+keyReleased(int key) : void	Taste key wurde losgelassen.

5.7.4 Klasse PDToGUIInterface (Singleton)

Über diese Schnittstelle sendet die PD die Änderungen des Spielfeldes an das User Interface. (Diese Klasse wird vom GUI implementiert.)

Funktionen

+movePlayer(int tox, int toy) : void	Die Spielfigur wird an die Koordinaten (tox,toy) verschoben.

5.7.5 Klasse PDToNETInterface

(vorläufig durch Netzwerküberbrückung ersetzt, bis Netzwerk implementiert ist)

5.7.6 Klasse NETToPDInterface

(vorläufig durch Netzwerküberbrückung ersetzt, bis Netzwerk implementiert ist)

5.7.7 Klasse Spielfeld

Die Klasse Spielfeld enthällt ein Array das den aktuellen Zustand und die Positionen aller Spielelemente representiert. Sie hat zugriff auf alle Spielentscheidenden Informationen.

Attribute

-feld : array of Spielelement*	Abbild des aktuellen Spielstandes. Jeder Eintrag im Array entspricht einem Feld. d.h. es können nicht mehrere Elemente auf einem Feld sein. (Spielfiguren werden hier nicht gespeichert!)
-playerlist : vector	Zeigerliste auf alle Spielfiguren des aktuellen Spiels. Die Position der Figur ist bei der Figur gespeichert.

Funktionen

+setField(int x, int y, int item) : void	Setzt ein bestimmtes Element an die Koordinate (x,y).
+getField(int x, int y) : int item	Liefert das Element das sich an der Koordinate (x,y) befindet.
+setPlayer(Spielfigur* player) : void	Meldet einen neuen Spielfigur beim Spielfeld an.

5.7.8 Klasse Spielfigur

Die Klasse Spielfigur enthällt alle wichtigen Informationen über Position und Zustand der Spielfigur.

Attribute

-position : Position	Position der Spielfigur auf dem Spielfeld in (x,y) Koordinaten.
-direction : int	Richtung in die die Spielfigur gehen mchte. (STAY, UP, DOWN, LEFT, RIGHT)
-numberOfBombs : int	Die Anzahl Bomben die er noch legen darf. -1 wenn Bombe gelegt wurde, +1 wenn seine Bombe explodiert ist oder ein Bomben-Powerup aufgenommen wurde.
-reichweite : int	Reichweite der Bombe in Feldern. Wird der Bombe übergeben wenn sie gelegt wird. Wird erhöht wenn ein Flammen-Powerup aufgenommen wird.
-alive : bool	TRUE wenn die Spielfigur noch lebt.
-bombPressed : bool	TRUE wenn die Bomben-lege-Taste gedrückt ist.

Funktionen

+setDirection(int dir) : void	Setzt die Laufrichtung der Spielfigur (STAY, UP, DOWN, LEFT, RIGHT).
+setBombPressed(bool bomb) : void	Setzt das bombPressd Flag (bomben-lege-Taste gedrückt).
+getPosition() : Position pos	Liefert die aktuelle Position der Spielfigur.
+addBomb() : void	Fügt eine Bombe zum Arsenal der Spielfigur hinzu.
+die() : void	Zerstört die Spielfigur wenn sie gesprengt wurde.
+setNetInterface(PD2NET*) : void	Setzt das Interface an das die Änderungen der Position geschickt werden müssen.
-move() : void	Bewegt die Spielfigur um ein Feld in die aktuelle Richtung (direction).
-dropBomb() : void	Legt eine Bombe an der aktuellen Position sofern noch Bomben im Arsenal.
-checkField() : bool ok	Prüft ob ein Feld auf dem Spielfeld passierbar ist oder nicht.

5.7.9 Klasse SpielerClient

Die Klasse Spieler empfängt die Benutzereingaben, bestimmt die daraus folgenden Aktionen und leitet sie an den Server weiter.

Attribute

-keylist : array of bool	Speichert die Informationen welche Tasten momentan gedrückt sind und welche nicht.

Funktionen

+keyPressed(int key) : void	Taste key wurde gedrückt.
+keyReleased(int key) : void	Taste key wurde losgelassen.
+setNetInterface(PD2NETI) : void	Setzt das Interface an das die Änderungen der Benutzereingaben geschickt werden müssen.
+getGUIInterface() : GUI2PD*	Liefert die Schnittstelle zur benutzereingabe.
-computeDirection() : void	Berechnet die Laufrichtung aus den Daten der keylist und sendet sie an den Server.

5.7.10 Klasse SpielfeldClient

Die Klasse SpielfeldClient hat ein vereinfachtes Abbild der Spielsituation gespeichert. Sie benötigt dieses später zur berechnung der Explosionen. Sie empfängt alle Änderungen vom Server und gibt sie ans GUI weiter.

Attribute

-field : array of int	Vereinfachtes Abbild der Spielsituation.
-players : array of Pos	Die Positionen der Spieler auf dem Spielfeld.

Funktionen

+setField(int x, int y, int item) : void	Setzt ein bestimmtes Element an die Koordinate (x,y).
+setPlayer(int x, int y, int nr) : void	Setzt die Spielfigur (nr) an die Position (x,y).
+getNetInterface() : NET2PD*	Liefert das Interface auf das die Änderungen auf dem Spielfeld geschickt werden müssen.
+setGUIInterface(PD2GUI*) : void	Setzt die Schnittstelle zur Spielfeldanzeige.

5.7.11 Klasse SpielElement

Abstrakte Klasse von der alle Objekte abgeleitet sind die auf dem Spielfeld platziert werden können. (ausgenommen Spielfiguren)
(Spielelement werden im 1. Prototyp noch nicht eingesetzt.)

Teil II

Iteration 2

Kapitel 6

Anforderungsspezifikation

6.1 Einführung

6.1.1 Zweck

Dieses Kapitel legt die Anforderungen an das Programm NetBomb fest.

6.1.2 Gültigkeitsbereich

Semesterarbeit Software Engineering, Sommersemester 2002

6.1.3 Definitionen, Akronyme, Abkürzungen

Siehe Anhang A auf Seite 88.

6.2 Allgemeine Beschreibung

6.2.1 Spielregeln

Ziel des Spiels ist es, die gegnerische Spielfigur mittels einer Bombe und dessen Bombenstrahl zu eliminieren. Im Spiel gibt es zwei Spielfiguren, eine fixe Anzahl Mauern und eine variable Anzahl Wände. Die Spielfigur kann Wände sprengen, Mauern sind unzerstörbar. Die Spielfigur kann weder durch Wände noch durch Mauern gehen. Unter gesprengten Wänden können sich Bomben oder Feuersymbole befinden. Das Aufheben einer Bombe erlaubt der Spielfigur das Legen einer zusätzlichen Bombe in Serie. Das Aufheben einer Flamme erlaubt der Spielfigur einen um ein Feld längerer Bombenstrahl. Die zwei Spielfiguren sind gegenseitig transparent. Sie können sich kreuzen.

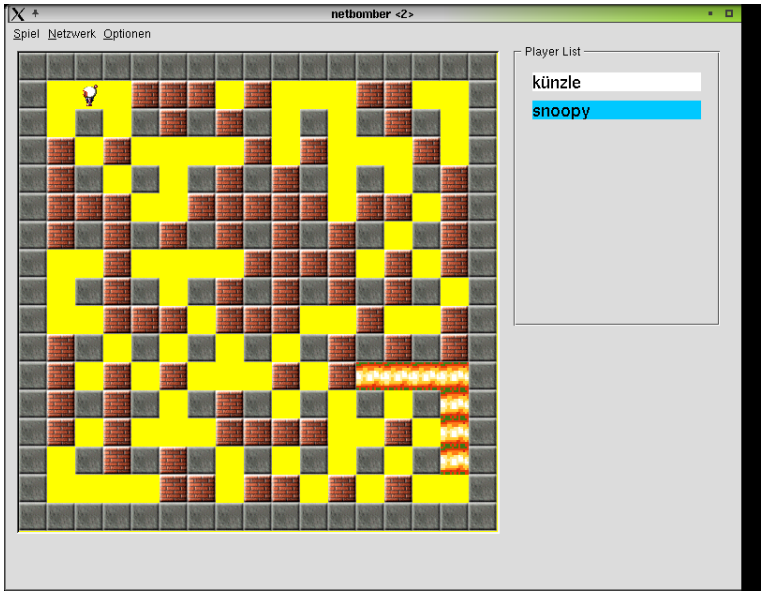

Abbildung 6.1: Bomberman Spiel

6.2.2 Benutzergruppen

Alle Menschen ob jung oder alt, weiblich oder männlich, die Freude an Computerspielen haben.

6.2.3 Mögliche Erweiterungen

- Sound Unterstützung
- spezielle Powerup-Icons, die von der Spielfigur aufgenommen werden können.
- Highscore Anzeige

6.2.4 Zu erwartende Probleme

wurden bereits im Dokument Projektplan eingetragen.

6.2.5 Annahmen

Da die von uns verwendeten Technologien für uns neu sind, sind wir uns bewusst, dass das Risiko vorhanden ist, dass wir die Anforderungen nicht vollständig erfüllen können. Dieses Dokument wurde in der Annahme geschrieben, dass wir die auftretenden Probleme lösen können. Ansonsten werden wir die Anforderungen zusammen mit dem Betreuer anpassen.

6.2.6 Abhängikeiten

Funktionalität Qt-Bibliothek und KDE-Bibliothek.

6.3 Anforderungen im Einzelnen

6.3.1 Funktionale Anforderungen Iteration 1

Spiel

Referenz	Funktion	Priorität
A1.1	Spiel starten	1
A1.2	Spiel beenden	1
A1.3	Spielregeln überwachen	1

Tabelle 6.1: Spiel Funktionen Iteration 1

Spielfigur

Referenz	Funktion	Priorität
A2.1	Figur bewegen	1

Tabelle 6.2: Spielfigur Funktionen Iteration 1

Spielfeld

Referenz	Funktion	Priorität
A3.1	Hintergrund zeichnen	1
A3.2	Mauern und Wände zeichnen	1
A3.3	Spielfigur zeichnen	1

Tabelle 6.3: Spielfeld Funktionen Iteration 1

6.3.2 Funktionale Anforderungen Ieration 2

Spiel

Referenz	Funktion	Priorität
A1.4	Spielstand aktualisieren	2
A1.5	Highscore speichern	2

Tabelle 6.4: Funktionen Spiel Iteration 2

Spielfigur

Referenz	Funktion	Priorität
A2.2	Bombe legen	2
A2.3	sterben	2

Tabelle 6.5: Funktionen Spielfigur Iteration 2

Spielfeld

Referenz	Funktion	Priorität
A3.4	Wand entfernen	2

Tabelle 6.6: Funktionen Spielfeld Iteration 2

Bombe

Referenz	Funktion	Priorität
A4.1	explodieren	2
A4.2	Reichweite berechnen	2

Tabelle 6.7: Funktionen Bombe Iteration 2

Netzwerk

Referenz	Funktion	Priorität
A5.1	Server starten	2
A5.2	Client anmelden	2
A5.3	Spielelement Position übermitteln	2
A5.4	Spielsituation synchronisieren	2

Tabelle 6.8: Funktionen Netzwerk Iteration 2

6.3.3 Use Cases Iteration 1

UC01 Spiel starten

Auslösender Aktor	Spieler
Zweck / Ziel	Spielfeld und Spielelemente zeichnen, Netzwerkverbindung aufbauen
Priorität	1
Style	casual
Anforderungen	Iteration 1: A1.1, A1.2, A3.1, A3.2, A3.3 Iteration 2: inkl.A5.1, A5.2, A5.3, A5.4
Vorbedingung	-
Nachbedingung	Spielfeld und Spielelemente gezeichnet, Netzwerkverbindung aufgebaut.
Bemerkungen	-

Tabelle 6.9: UC01 Spiel starten

Grundlegender Ablauf

Aktor	**System**
1. Benutzer startet neues Spiel	
	2. Leveldaten einlesen
	3. Spielfeld zeichnen
	4. Spielelemente zeichnen
	5. wartet auf Benutzereingabe

Erweiterungen
∗a zu jeder Zeit kann der Spieler das Spiel beenden

UC02 Spielfigur bewegen

Auslösender Aktor	Spieler
Zweck / Ziel	Aktor kann Spielfigur in horizontaler oder vertikaler Richtung bewegen
Priorität	1
Style	casual
Zu erfüllende Anforderungen	A1.3, A2.1, A3.3
Vorbedingungen	UC01
Nachbedingungen	Die Spielfigur wurde um ein Feld verschoben.
Bemerkungen	Dieser UC kann 1 oder n mal ausgeführt werden.

Tabelle 6.10: UC02 Spielfigur bewegen

Grundlegender Ablauf

Aktor	**System**
1. Der Aktor verschiebt die Spielfigur um ein Feld nach links, rechts, oben oder unten.	
	2. zeichnet die Figur auf dem neuen Feld, sofern das Zielfeld nicht einer Wand oder einer Mauer enstpricht.

6.3.4 Use Cases Iteration 2

UC03 Bombe legen

Auslösender Aktor	Spieler
Zweck / Ziel	Der Spieler legt eine Bombe, die nach einer gewissen Zeit explodiert und alle vernichtbaren Objekte, die sich in Reichweite befinden zerstört
Priorität	2
Style	casual
Anforderungen	A1.1, A1.3, A2.1, A2.2, A2.3, A3.1, A3.2, A3.3, A3.4, A4.1, A4.2
Vorbedingung	UC01
Nachbedingung	Objekte die sich innerhalb der Reichweite befunden haben sind zerstört.
Bemerkungen	-

Tabelle 6.11: UC03 Bombe legen

Grundlegender Ablauf

Aktor	**System**
1. Der Spieler legt eine Bombe	
2. Der Spieler bewegt sich auf ein anderes Feld	
	3. Auf dem alten Feld wird eine Bombe dargestellt
	4. Ein Timer startet
	5. Der Timer ist abgelaufen, auf allen Feldern (horizontal und vertikal zur Bombe) in Reichweite wird eine Explosion dargestellt
	6. Zerstörbare Elemente, die sich auf diesen Feldern befunden haben werden zerstört

Erweiterungen

∗a zu jeder Zeit kann sich der Spieler auf benachbarte, begehbare Felder bewegen

6.3.5 Optional

Spiel

Referenz	Funktion	Priorität
A1.6	Spiel pausieren	3
A1.7	Soundeffekte abspielen	3
A1.8	Musik abspielen	3

Tabelle 6.12: optionale Funktionen Spiel

Spielfigur

Referenz	Funktion	Priorität
A2.4	Bombe-Powerup aufnehmen	3
A2.5	Flamme-Powerup aufnehmen	3

Tabelle 6.13: optionale Funktionen Spielfigur

Spieloptionen

Referenz	Funktion	Priorität
A6.1	Sound ein/aus	3
A6.2	Highscores anzeigen	3
A6.3	Spielername eingeben	3

Tabelle 6.14: Spieloptionen

6.3.6 Leistungs- und Mengenanforderungen

Leistungsanforderungen

Um die Spielbarkeit übers Netzwerk zu gewährleisten, muss der Spielstatus von allen Spielern mindestens alle 150ms synchronisiert werden.

Mengenanforderungen

Keine.

6.3.7 Anforderungen an Schnittstellen

Benutzerschnittstelle

Das System ist mit dem Keyboard und der Maus bedienbar.

Software Schnittstellen

Qt, KDE-Library

6.3.8 Randbedingungen für den Entwurf

Übereinstimmungen mit Normen

SE01/02

Einschränkungen bezüglich Software

Lauffähig unter KDE 2.2 mit Qt 2.3.1

Einschränkungen bezüglich Hardware

Lauffähig unter allen UNIX-Derivaten, die KDE unterstützen. Die Leistungsanforderungen gelten für ein Netzwerk (mind. 10Mb/s) ohne zusätzlichen Datenverkehr.

6.3.9 Merkmale

Benutzbarkeit

Die Bedienung des Programms entspricht den gängigen KDE-Programmen:
http://developer.kde.org/documentation/standards/kde/style/basics/index.html

6.3.10 Andere Anforderungen

Inbetriebnahme / Installation

Standardinstallationsweg eines Linux-Quellcodes (configure, make, make install) In der Datei README finden Sie bezüglich Inbetriebnahme detaillierte Informationen.

Konfigurierbarkeit

Alle Konfigurationen werden gespeichert. Die IP-Adresse des Spielservers kann eingegeben werden. Optional kann der Spielername eingegeben werden.

Kapitel 7

Supplementary Specification

7.1 Functionality

7.1.1 Error handling

Fehler werden dem Spieler mittels einer Fehlermeldung gemeldet. Die Fehler werden nicht persistent in einer log-datei gespeichert.

7.1.2 Security

Das Programm hat keine speziellen Sicherheitsvorkehrungen. Es kommuniziert über ein IP-Netz mit dem Programm der anderen Spieler.

7.2 Performance

Mit den unter 6.3.6 auf Seite 39 angegebenen Leistungsanforderungen soll ein angenehmes Spielverhalten gewährleistet werden.

7.3 Supportability

7.4 Free open source components

Als Linux-Software-Developers liegt es Nahe, OpenSource Technologien zu verwenden. Das Programm läuft unter der GPL (General Public License). Mehr dazu unter http://www.gnu.org

7.5 Developer Guidelines

7.5.1 Code Guidelines

- Methodennamen beginnen mit Kleinbuchstaben
- Klassennamen beginnen mit Grossbuchstaben
- Variabeln beginnen mit Kleinbuchstaben
- Vor und nach Operations-Zeichen wird ein Abstand gemacht.
- Nach zusammengehörigen Code-Blöcken eine leere Zeile einfügen.

- Bei Klassen die öffnende geschweifte Klammer rechts neben dem Klassennamen und die schliessende auf eine eigene Zeile. Für alle anderen Fälle die öffnende und schliessende geschweifte Klammer auf eine eigene Zeile.
- Code innerhalb geschweifter Klammern wird eingerückt.
- Falls Tabulatoren verwendet werden, in der Entwicklungsumgebung definieren, dass dafür Leerzeichen eingefügt werden.
- Default-Einrückung: Zwei Leerzeichen
- jede Kommentarzeile mit zwei slashes (//) beginnen.

Kapitel 8

Domainanalyse

8.1 Konzeptionelles Modell

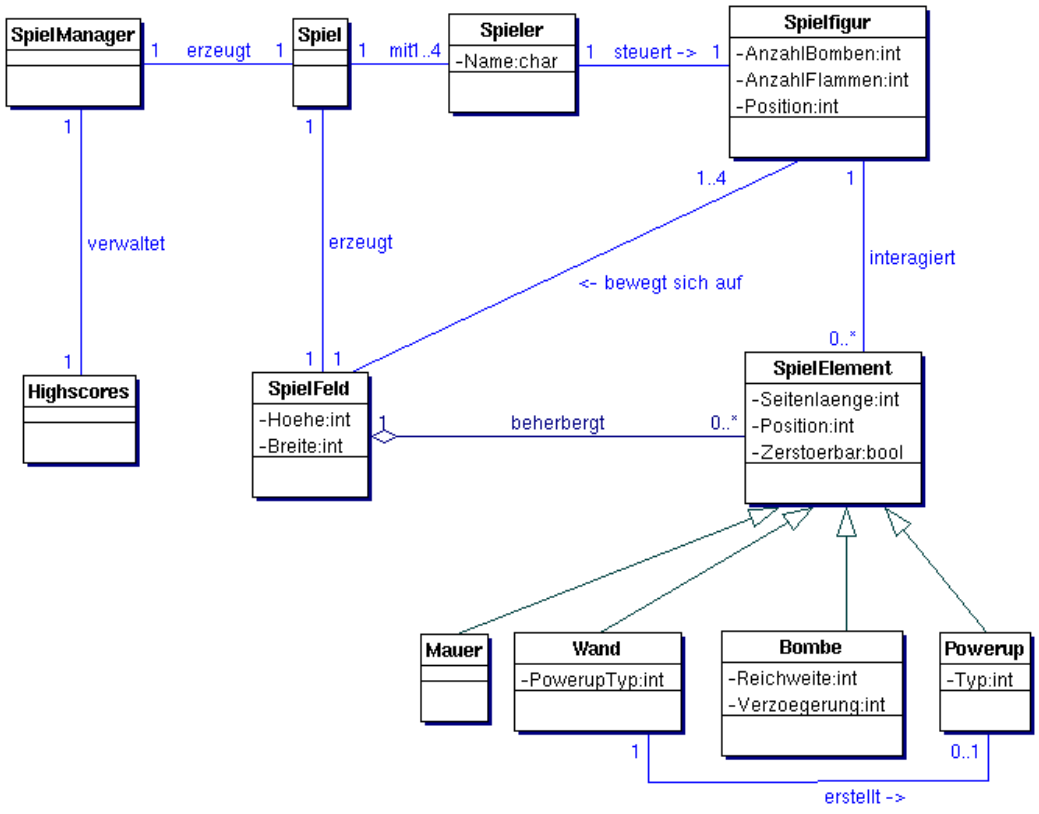

Abbildung 8.1: Domainmodell

8.1.1 Spielmanager

Der SpielManager ist für die Initialisierung des Spiels zuständig. Er kennt alle Spieler und weitere für das Spiel notwendige Parameter.

8.1.2 Spiel

Das Spiel erzeugt das Spielfeld und positioniert die Spielelemente für den Start des Spiels.

8.1.3 Spielfeld

Das Spielfeld ist der Hintergrund auf dem das eigentliche Spiel stattfindet. Es hat eine bestimmte Höhe und Breite die zu Beginn festgelegt werden. Auf dem Spielfeld befinden sich die Objekte des Spielfeldes: Spielelemente und Spielfiguren. Alle Objekte des Spielfeldes haben eine Position. Das Spielfeld kennt alle Objekte und ihre Positionen.

8.1.4 Spielelement

Das Spielelement dient als Oberklasse für sämtliche sich auf dem Spielfeld befindlichen Objekte, ausser der Spielfigur. Die einheitliche Schnittstelle erleichtert die Realisierung einiger Funktionen wie die Zerstörung von Objekten usw. Ein Objekt des Spielfeldes belegt normalerweise ein Feld auf dem Spielfeld. Es kann über das Spielfeld die Belegung der benachbarten Felder abfragen.

8.1.5 Spielfigur

Die Spielfigur ist das einzige Objekt des Spielfeldes das sich auf dem Spielfeld bewegen kann. Sie kann mit den Spielelementen kommunizieren. Die Spielfigur kann zum Beispiel Bomben erzeugen (legen) oder Powerups aufnehmen. Andererseits wird sie von Mauern und Wänden am weitergehen gehindert.

8.1.6 Mauer

Die Mauer wird vor Spielbeginn erzeugt und bleibt während des ganzen Spiels an ihrem Platz. Sie kann nicht zerstört werden, trotzt also auch explodierenden Bomben.

8.1.7 Wand

Die Wand wird ebenfalls zu Beginn erzeugt, kann aber durch Bomben gesprengt werden und verschwindet in diesem Fall vom Spielfeld. Eine Wand kann (muss aber nicht) ein Powerup beherbergen. Dieses bleibt auf dem Feld liegen falls die Wand zerstört wird.

8.1.8 Bombe

Die Bombe wird von der Spielfigur erzeugt. Dabei werden die Attribute wie Reichweite und Verzögerungszeit gesetzt. Danach ist die Bombe ein eigenständiges Spielelement und explodiert entweder nach Ablauf der Verzögerungszeit oder wenn sie von einer anderen Bombe gesprengt wird. Nach dem Explodieren meldet sie ihr Ableben der Spielfigur, die sie erzeugt hat.

8.1.9 Powerup

Ein Powerup ist entweder vom Typ Bombe (eine Bombe mehr in Serie) oder vom Typ Flamme (grössere Reichweit) und ist zu Beginn unter einer Wand verborgen. Es tritt erst in Erscheinung wenn die Wand durch eine Bombe zerstört wurde. Betritt eine Spielfigur dasselbe Feld, nimmt sie das Powerup auf und es wird gelöscht. Wird das Powerup von einer explodierenden Bombe erfasst, wird es zerstört.

8.1.10 Spieler

Der Spieler hat einen Namen und steuert die Spielfigur. Er kann sie in alle 4 Richtungen bewegen und Bomben legen.

Kapitel 9

Software Architektur

9.1 Logische Sicht

Das System besteht aus primär 4 Schichten (3 Schichten - Architektur plus Netzwerk). Diese beinhalten:

Schicht 1 Graphisches User Interface (GUI)

Schicht 2 Problem Domain (PD)

Schicht 3 Datenhaltung (DH)

Schicht 4 Netzwerk (bestehend aus Client und Server)

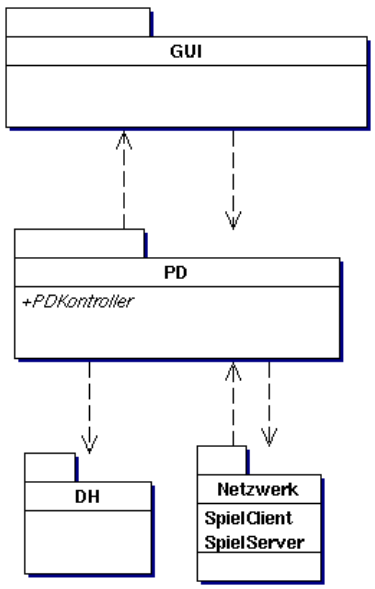

Abbildung 9.1: Schichtenmodell

Diese Architektur haben wir gewählt um eine klare Trennung zwischen den einzelnen Programmteilen zu erreichen.

Das graphische User Interface ist die Schnittstelle zum Benutzer. Über dieses kann er Eingaben machen oder in unserem Fall erfolgt die Steuerung der Spielfigur über das GUI.
Die PD ist die Schicht zwischen GUI und DH. Das heisst sie bekommt und verarbeitet Befehle vom GUI, schreibt Daten in die DH und ruft Funktionen in der Netzwerkschicht auf.
Die Datenhaltung ist dafür zuständig, Daten, die konsistent sein müssen zu speichern, damit sie bei einem Neustarten des Spiels wieder zur Verfügung stehen.
Die Netzwerkschicht regelt die Datenübertragung zwischen dem Server und den Clients.

9.2 Netzwerk

Das Netzwerk besteht aus einem Server und bis zu vier Clients die miteinander kommunizieren. Dabei kann jeder Client auch Server sein, das heisst zu Beginn des Spiels entscheidet der Spieler, ob er Server und Client oder nur Client ist. Es kann nur ein Spieler Server sein. Ist ein Spieler Server und Client, werden die Daten der Mitspieler zu ihm übermittelt. Das geschieht folgendermassen:
Die aktuellen Bewegungen eines Clients, also eines Spielers, werden zum Server übermittelt. Diese Daten werden vom Server entgegengenommen. Dieser berechnet damit die aktuelle Position und allfällige Aktionen des Clients. Diese berechnete Postion schickt der Server dann *allen* Clients zurück. Das heisst, jeder Client bekommt vom Server einen Snapshot. Zusätzlich zur Position beinhaltet dieser wichtige Daten wie zum Beispiel ob eine Bombe gelegt worden ist oder ein Powerup aufgenommen wurde. Mit diesen Angaben berechnet der Client selbständig, was auf dem Spielfeld passiert. Er macht also dieselben Berechnungen wie der Server. Falls ein Spielelement gelöscht werden muss, versucht der Client das zu machen. Da er aber nicht selbständig Elemente löschen darf, wartet er auf den Befehl des Servers, das entsprechende Element zu löschen. Der Client führt also nur eine Art Dummy-Funktion aus. Damit erreichen wir eine bessere Performace wie mit dem Prinzip, bei dem alle Daten vom Server zum Client geschickt werden und zudem können wir damit sicherstellen, dass alle Clients den selben Spielstand haben. Die Verbindung läuft über TCP, was das Ankommen der Pakete sicherstellt.
Der Server wurde mit dem Reactor Pattern implementiert. Dieses wird nachfolgend noch genauer erklärt.

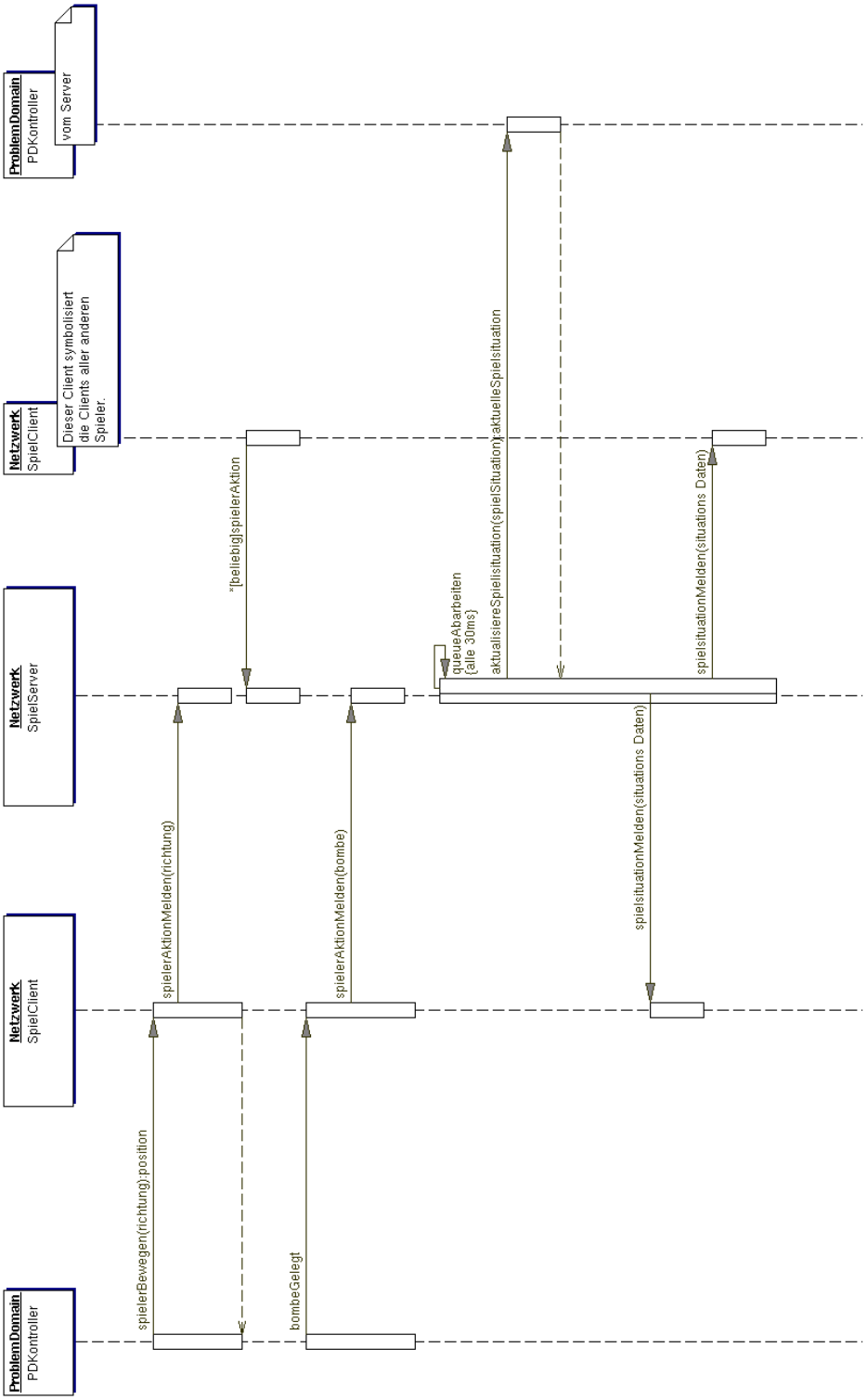

Abbildung 9.2: Interaktionsmodell Netzwerk

Kapitel 10

Designmodell

10.1 GUI Externes Design

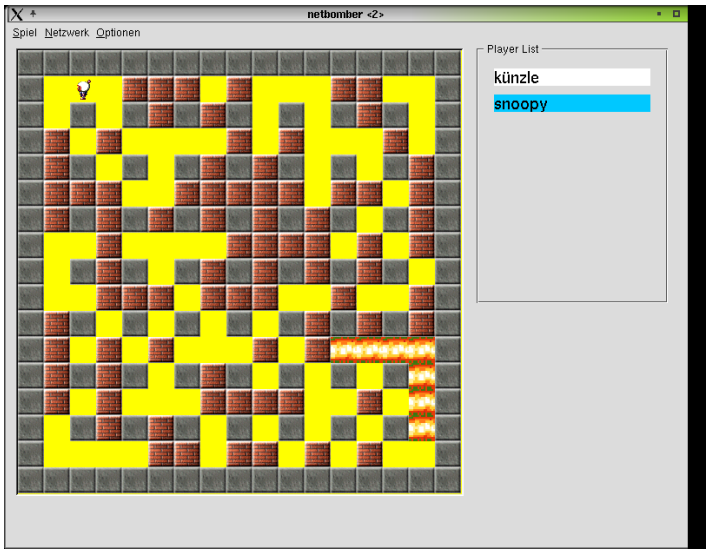

Abbildung 10.1: Endversion des GUIs

Das Spielfeld (links) besteht aus diversen Spielelementen. Diese werden dynamisch geladen, je

nach dem was der Server verlangt zum Darstellen. Rechts vom Spielfeld befindet sich die Playerlist. Jeder Spielername wurde unterschiedlich eingefärbt, genau so wie die Farbe des entsprechenden Kopfes der Spielfigur.

Die Farben der Spielfiguren:

Spielfigur 1	Weiss	RGB (255, 255, 255)
Spielfigur 2	Blau	RGB (0, 200, 255)
Spielfigur 3	Grün	RGB (60, 255, 0)
Spielfigur 4	Violet	RGB (255, 60, 255)

Tabelle 10.1: Farben der Spielfiguren

Abbildung 10.2: Snapshot des Dialoges Server anmelden

Dieser Dialog dient zum Anmelden an einen Server.

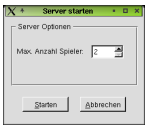

Abbildung 10.3: Snapshot des Dialoges Server starten

Dieser Dialog dient zum Starten eines Servers.

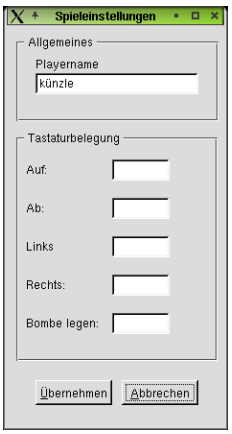

Abbildung 10.4: Snapshot des Dialoges Einstellungen

Dieser Dialog dient primär zum Konfigurieren des Spielernamens. Leider reichte uns die Zeit nicht mehr, die Tastaturbelegung variierend zu machen. Vorgesehen war es jedoch in diesem Dialog.

10.2 GUI Klassendiagramm

Das gesamte Klassendiagramm befindet sich im Register Nr. 9.

10.3 GUI Sequenzdiagramme

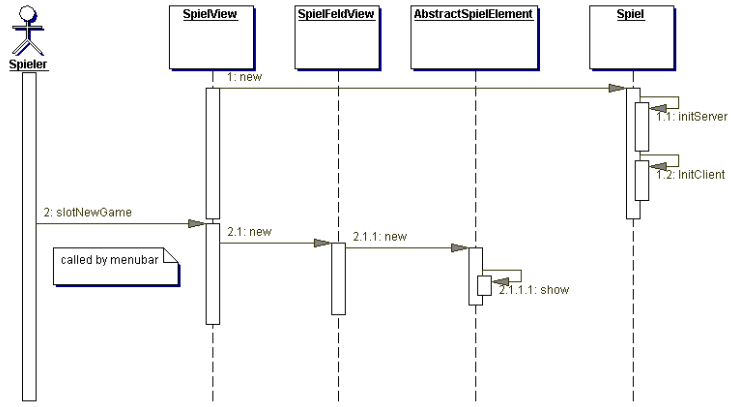

Abbildung 10.5: Sequenzdiagramm für neues Spiel

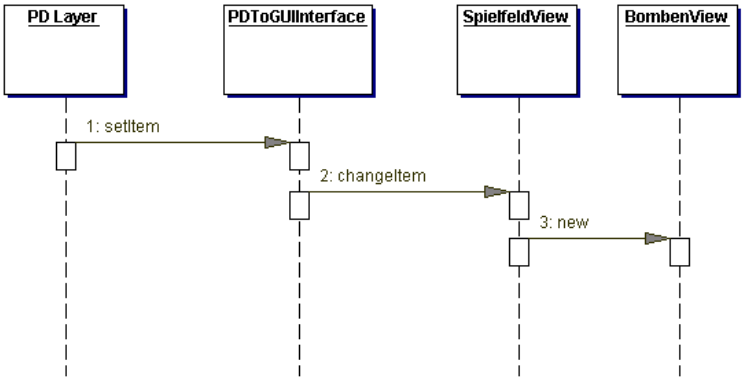

Abbildung 10.6: Sequenzdiagramm für Bombe legen

Die PD wird hier als Blackbox betrachtet. Diese ruft über das PDToGUIInterface setItem auf, wobei setItem als Parameter den typ BOMBE und die Koordinaten x und y beinhaltet. Die PDToGUIInterface Instanz teilt diese Anforderung der SpielfeldView Instanz mit, welche dann mit dem Erzeugen einer neuen BombenView Instanz reagiert.

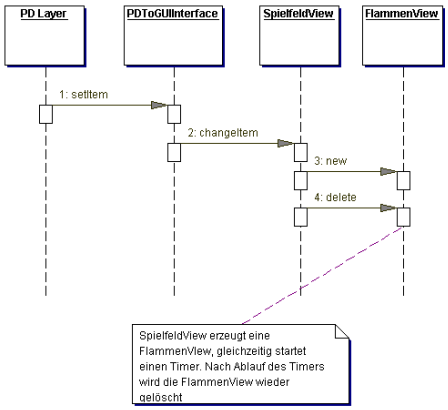

Abbildung 10.7: Sequenzdiagramm für eine Flamme anzeigen

Wieder vom PD Layer her kommt die Anforderung mit einem setItem. Diesmal mit den Parametern FLAMME als typ und die Koordinaten x und y. Die PDToGUIInterface Instanz leitet die Anforderung an die Instanz von SpielfeldView weiter. Die SpielfeldView Instanz erzeugt danach eine neue FlammenView Instanz. Gleichzeitig startet einen Timer (QTimer), welcher nach Ablauf den Slot timerDone() aufruft. In timerDone() wird die FlammenView wieder zerstört. Das bewirkt im Spiel selber das kurze Erscheinen der Flamme, sozusagen explosionsartig.

10.4 GUI Klassenbeschreibung

10.4.1 Klasse SpielView

Die Klasse SpielView enthält einzig allein ein Menu. Über dieses kann der Spieler einen Server starten, sich bei einem Server anmelden, Einstellungen vornehmen und last but not least das Spiel starten bezw. beenden. Um dies zu ermöglichen, hat die Klasse SpielView Pointers auf die Klassen JoinServerView, ServerView und SpielOptView. Diese werden bei den entsprechenden Slots (Callbacks des Menus), mit show() aufgerufen. Diese Klasse assoziiert ebenfalls mit der Klasse SpielFeldView, sozusagen der Hauptklasse des GUIs (siehe unten). Im Konstruktur wird diese erzeugt, jedoch noch nicht angezeigt. Dies erfolgt erst nachdem der Menupunkt Spiel starten ausgewählt wurde.

10.4.2 Klasse SpielFeldView

Sie ist sozusagen die Kernklasse im GUI Design. Diese Klasse verwaltet alle sich auf dem Spielfeld befindenen Elementen (Aggregation zu AbstractSpielElementView). Um das zu erreichen, wird sie von der Klasse QCanvasView abgeleitet. QCanvasView ist die Präsentationsklasse für QCanvasItems, also für einzelne Grafikelementen. Auch das Keyboard-Eventhandling erfolgt in dieser Klasse. Falls Keyboard-Events auftreten, werden diese abgefangen, und nur die für das Spiel relevanten Steuersignale werden der Klasse PDToGUIInterface weitergeleitet, wo sie dann von der PD verarbeitet werden. Gleichzeitig erhält sie von der Klasse GUIToPDInterface Methodenaufrufe, welche das Verändern des UI's zur Folge haben. Dies beinhaltet Items hinzufügen und entfernen (Bombe und Flamme), die Richtung der Spielfigur festlegen (um das entsprechende Sprite zu laden) und die Figur an eine andere Koordinate bewegen. Um die Spielfigur eindeutig zu identifizieren, hat die Methode moveXY() zusätzlich den Paramter ID, dessen Wert die PD weiss. Diese Klasse ist

fast intelligenzlos. Sie nimmt nur Befehle von der Interfaceklasse entgegen, oder leitet an diese Steuersignale weiter.

10.4.3 Klasse SpielfigurView

Diese Klasse repräsentiert die Spielfigur. Sie wird im Konstruktor der Klasse SpielFeldView erzeugt. Ihre Koordinaten erhält sie ebenfalls von der Klasse SpielFeldView. Um grafisch ansprechende Animationen zu ermöglichen, ist sie von der Klasse QCanvasSprite abgeleitet. Diese Klasse ermöglicht das dynamische Laden von Bildern (Frames), welche durch entsprechende Methoden zu einer Animation zusammengesetzt werden können. In der Methode advance(int step) erfolgt die eigentliche Animation in zwei Schritten: Wenn step 0 ist, hat man die Möglichkeit, die Items auf dem Spielfeld auf Kollision zu überprüfen (collision detection). Im zweiten Schritt (step ist 1) werden die Items, in unserem Fall die Spielfigur, bewegt. Diese Methode wird von Qt aufgerufen, vorausgesetzt das setAnimated(true) aiufgerufen wurde.

10.4.4 Klasse AbstractSpielElementView

Diese Klasse wurde von der Klasse QCanvasPolygonalItem abgeleitet. Sie ergänzt diese Klasse nur mit den Koordinaten. Gleichzeitig dient sie als Oberklasse für alle auf dem Spielfeld befindenen Elementen, ausser der Spielfigur. Dies ist notwendig, um die Aggregation zwischen SpielFeldView und AbstractSpielElement zu erreichen.

10.4.5 Klasse MauerView

Sie repräsentiert auf dem Spielfeld eine unzerstörbare Mauer. Das Attribut rtti steht für run time type information. Sie dient zur Identifikation.

10.4.6 Klasse BombenView

Sie repräsentiert eine Bombe auf dem Spielfeld.

10.4.7 Klasse FlammenView

Sie repräsentiert eine Flamme auf dem Spielfeld.

10.4.8 Klasse ServerView

Diese Klasse repräsentiert einen Dialog, über welchen einen Server gestartet werden kann. Als Option kann die max. Anzahl Spieler festgelegt werden, nach dem Motto Server ist König. Die Anmeldung erfolgt via den Klassen SpielView und GUIToPDInterface zur PD mit dem Callback (Slot) bStartenPressed.

10.4.9 Klasse JoinServerView

Diese Klasse repräsentiert einen Dialog, über welchen sich ein Spieler an einen Server anmelden kann. Das einzige, was der Spieler tun muss, ist eine gültige IP-Adresse (bei welcher einen NetBomb Server gestartet wurde) eingeben. Der Anmeldevorgang beginnt nach dem Drücken des Anmelden-Buttons, bezw. in der Methode bStartenPressed.

10.4.10 Klasse SpielOptView

Sie repräsentiert einen Dialog, über welchen der Spieler Einstellungen zur Spielersteuerung und seinen Playername eingeben kann.

10.4.11 Klasse GUIToPDInterface

Sie ist die Kommunikations-Schnittstelle vom GUI zur PD. Es werden die für das Spiel relevanten Keyboard Events zur PD weitergeleitet mit keyPressed und keyReleased. Die PD erhält über die Schnittstelle die Aufforderung den Server zu starten mit startServer(). Um sich an einen Server anzumelden wird dir PD mit joinServer und den Parametern Spielername bezw. IP dies entsprechend mitgeteilt. Um das Spiel schlussendlich zu starten, wird der PD mit startGame() migeteilt. Wir wählten das Singleton Pattern der GoF, damit es einerseits nur einmal instanziert ist, andererseits von jedem beliebigen Ort zu Verfügung steht.

10.4.12 Klasse PDToGUIInterface

Über diese Klasse nimmt das GUI die Wünsche der PD entgegen. Alle Aufrufe gehen danach weiter an die Klasse SpielFeldView. Auch hier das Singleton Pattern der GoF aus den gleichen Gründen wie bereits erwähnt wurde.

10.5 PD Klassendiagramm

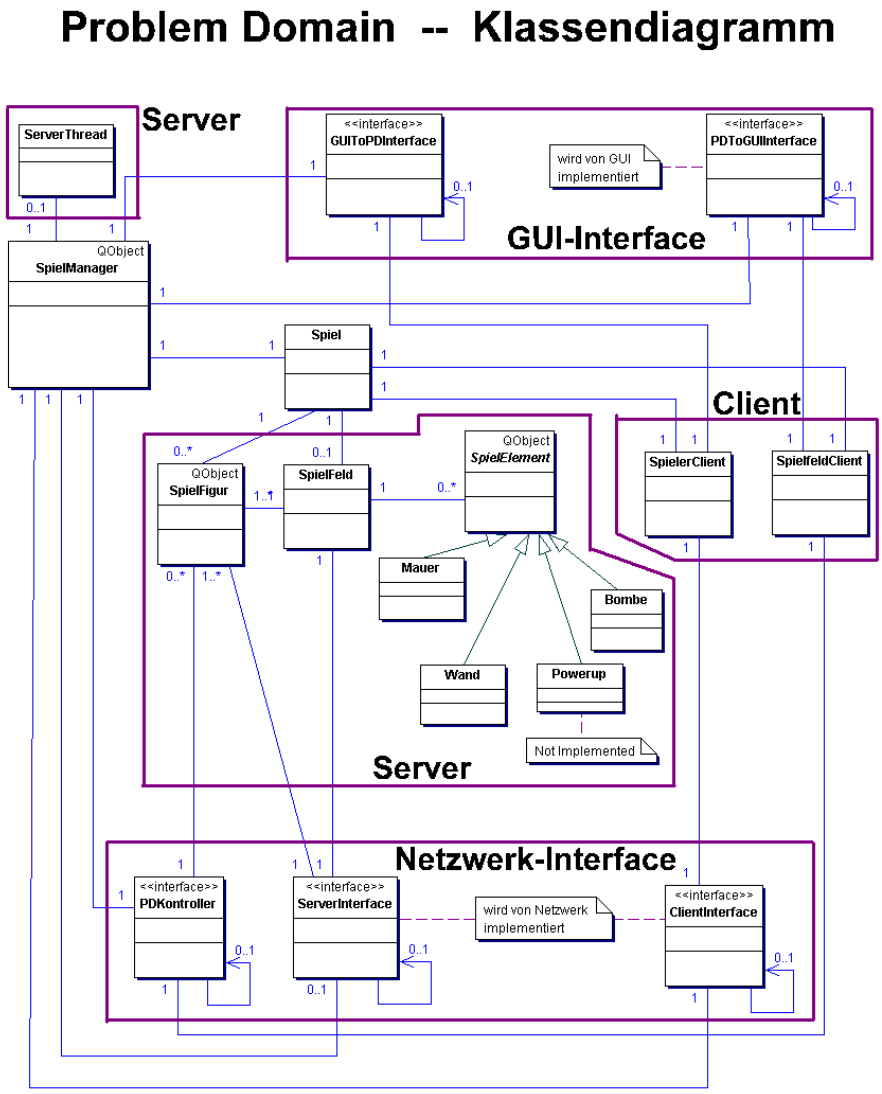

Abbildung 10.8: Klassendiagramm PD (siehe auch A3 Blatt in Register 9)

10.6 PD Sequenzdiagramme

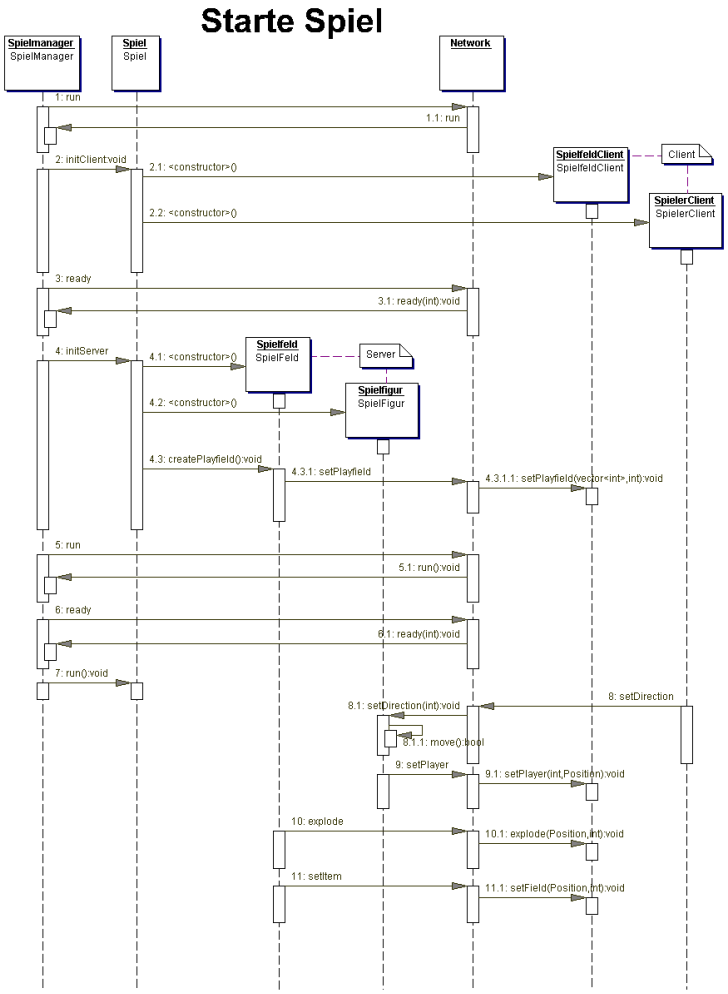

Abbildung 10.9: Sequenzdiagramm für neues Spiel

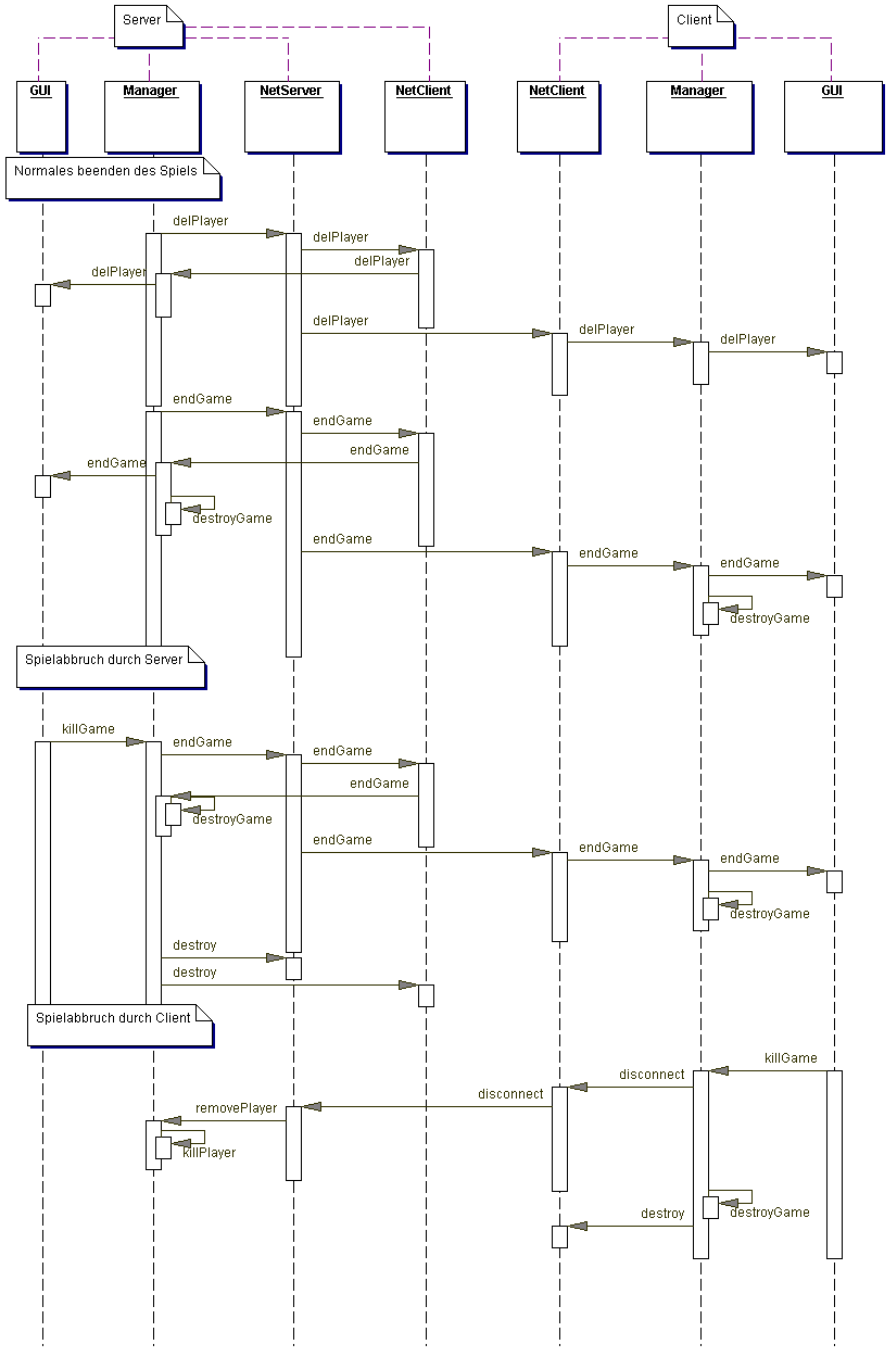

Abbildung 10.10: Sequenzdiagramm für das Beenden des Spiels

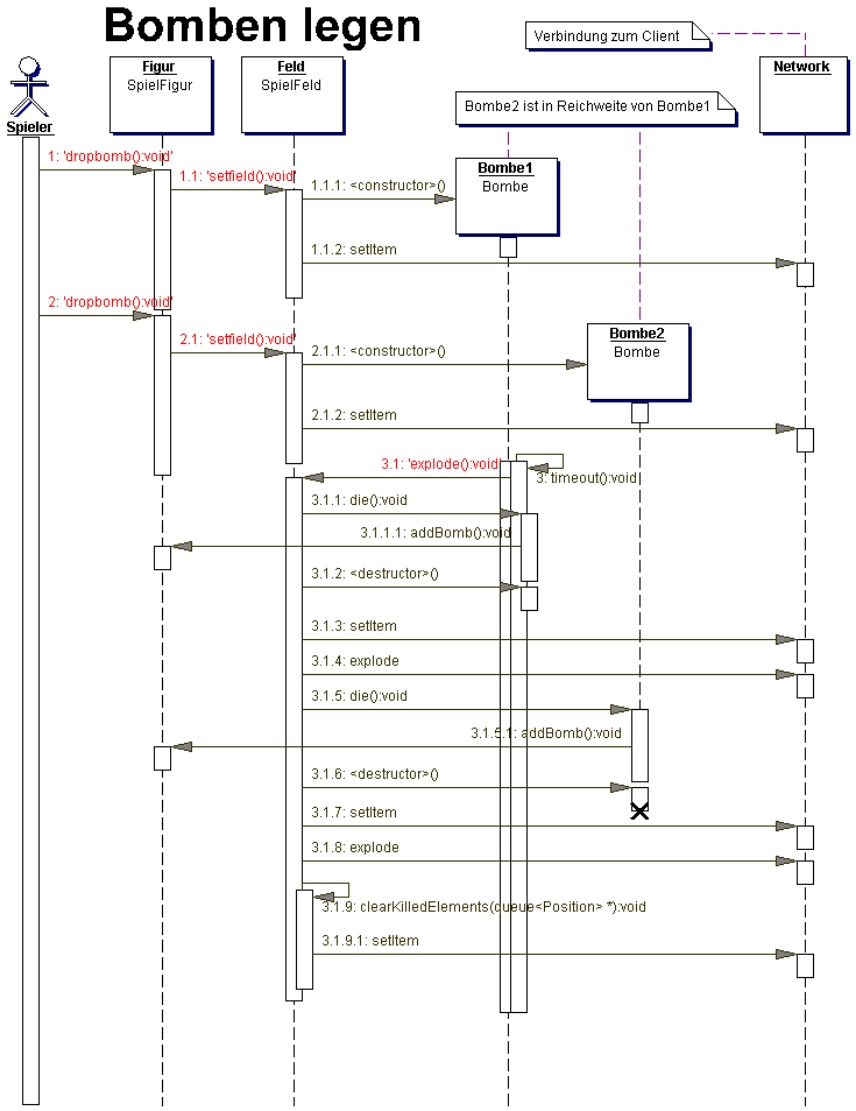

Abbildung 10.11: Sequenzdiagramm für das Legen von Bomben

10.7 PD Klassenbeschreibung

Die endgültige Version des Spiels soll über ein Netzwerk gespielt werden. Deshalb wird die PD in Server und Client unterteilt. Die Klassen des Servers sind für die Berechnung des gesamten Spiel-

verlaufs (Positionen, Treffererkennung, ...) verantwortlich. Der Client konvertiert die Steuerbefehle und schickt sie über das Netzwerk an den Server. Der Server berechnet die Auswirkungen und schickt die Änderungen zurück an alle Clients. Der Client reicht die Änderungen wiederum an das User Interface weiter. Jeder Spieler hat auf seinem Rechner eine Instanz des Clients, aber nur einer hat zusätzlich noch eine Instanz des Servers.

10.7.1 Klasse SpielManager

Der Spielmanager ist für die Verwaltung des Spiels verantwortlich. Er kontrolliert den Verbindungsaufbau und -abbruch der Clients und erzeugt die zum Spielen notwendigen Objekte.

Attribute

-serverGame : bool	True wenn dieser PC als Server agiert.
-playerConnected : bool[]	True wenn mit der entsprechenden ID ein Spieler verbunden ist.
-playerName : string[]	Namen der verbundenen Spieler.
-playerReady : bool[]	True wenn der Spieler zum Spielen bereit ist.
-game : Spiel*	Pointer auf das Spiel-Objekt.
-serverThread : ServerThread*	Pointer auf Server-Listening-Thread.
-pdMutex : QMutex*	Mutex zum Schutz der PD vor Mehrfachzugriff.
-serverMutex : QMutex*	Mutex zum Beenden des Server-Threads.
-client : Client*	Pointer auf Client-Objekt, zum Abfragen des Sockets.
-clientTimer : QTimer*	Timer zur regelmässigen Abfrage des Client-Sockets.
-guiInputInterface : GUIToPDInterface*	Interface GUI → PD
-guiOutputInterface : PDToGUIInterface*	Interface PD → GUI
-netInputInterface : PDKontroller*	Interface NET → PD
-netOutClientInterface : ClientInterface*	Interface PD (client) → NET
-netOutServerInterface : ServerInterface*	Interface PD (server) → NET

Funktionen

+startServer(string playerName) : void	Startet den Netzwerk-Server und meldet sich als Spieler an.
+joinServer(string ipAddress, string playerName) : void	Startet den Netzwerk-Client und meldet sich beim Server an.
+startGame() : void	Initialisiert und startet das Spiel mit den verbundenen Spielern.
+endGame() : void	Beendet das laufende Spiel.
+killGame() : void	Bricht das laufende Spiel ab. (zu irgendeinem Zeitpunkt)
+errorMsg(int msgNr, string addInfo) : void	Bearbeitet Fehlermeldungen vom Netzwerk.
+newPlayer(int playerID, string playerName) : void	Ein neuer Spieler hat sich beim Server angemeldet.
+removePlayer(int playerID) : void	Entfernt einen Spieler aus der Spielerliste und entfernt ihn aus dem Spiel.
+ready(int playerID) : void	Setzt den Spieler spielbereit.
+distributeMsg(string info) : void	Schickt eine Nachricht an alle verbundenen Clients weiter.
+connectConfirm() : void	Bestätigt die Verbindung zum Server.
+setPlayerName(int playerID, string name) : void	Setzt den Namen eines Verbundenen Spielers.
+disconnect() : void	Meldet sich beim Server ab.
+run() : void	Synchronisiert und startet das Spiel.
+infoMsg(string info) : void	Zeigt eine Nachricht vom Server an.
-createNetworkServer() : void	Startet den Server-Listening-Thread.
-killNetworkServer() : void	Beendet den Server-Listening-Thread.
-createNetworkClient() : void	Startet den Client-Timer zur Abfrage des Sockets.
-killNetworkClient() : void	Beendet den Client-Timer.
-clientTimeout() : void	wird bei Ablauf des Client-Timers aufgerufen.

10.7.2 Klasse Spiel

Die Klasse Spiel erzeugt die restliche Spielstruktur (Spielfeld, Spielfiguren) je nach Anzahl Spieler. Es wird immer ein Client erstellt, und beim Spielführer zusätzlich noch ein Server. Sie ist dafür verantwortlich, dass das Spiel mit allen Clients synchronisiert ist bevor das Spiel gestartet wird.

Attribute

-serverFeld : Spielfeld*	Pointer auf das ServerSpielfeld.
-figur[4] : Spielfigur*	Pointer auf die Spielfiguren des Servers.
-clientFeld : SpielfeldClient*	Pointer auf das ServerSpielfeld.
-spieler : SpielerClient*	Pointer auf die Spielfigur des Clients.

Funktionen

+initClient() : void	Initialisiert die Client-Umgebung.
+initServer(bool activPlayers[], string playerNames[]) : void	Initialisiert die Server-Umgebung.
+ready() : bool	gibt TRUE zurück wenn Client bereit ist.
+run() : void	Synchronisiert und startet das Spiel.
+killPlayer(int playerID) : void	Entfernt einen Spieler vom Spielfeld.
-destroyClient() : void	Räumt die Client-Umgebung nach Spielende auf.
-destroyServer() : void	Räumt die Server-Umgebung nach Spielende auf.

10.7.3 Klasse GUIToPDInterface (Singleton)

Über diese Schnittstelle sendet das User Interface die Tastatureingaben des Spielers an die PD.

Attribute

-manager : SpielManager*	Pointer auf den Spielmanager.
-spieler : SpielerClient*	Pointer auf den Spieler.
-pdMutex : QMutex*	Sichert die PD vor Mehrfachzugriff ab.

Funktionen

+setManager(SpielManager* man, QMutex* pdMut) : void	Registriert den Spielmanager.
+setPlayer(SpielerClient* player): void	Registriert den Spieler.
+startServer(const char* player) : void	Startet neues Spiel als Server.
+joinServer(const char* ipAdress, const char* playerName) : void	Startet neues Spiel als Client.
+startGame() : void	Startet das Spiel mit den verbundenen Spielern (nur Server)
+killGame() : void	Beendet das Spiel zu einem beliebigen Zeitpunkt.
+keyPressed(int key) : void	Eine Taste wurde gedrückt.
+keyReleased(int key) : void	Eine Taste wurde losgelassen.

10.7.4 Klasse PDToGUIInterface (Singleton)

Über diese Schnittstelle sendet die PD die Änderungen des Spielfeldes an das User Interface. (siehe GUI)

10.7.5 Klasse ServerInterface

(siehe Netzwerk)

10.7.6 Klasse ClientInterface

(siehe Netzwerk)

10.7.7 Klasse ServerThread

Separater Thread für die Abfrage des Server-Sockets. (Abgeleitet von QThread)

Attribute

-runServer : QMutex*	Synchronisationsobjekt zum Beenden des Serverthreads.
-pdMutex : QMutex*	Sichert die PD vor Mehrfachzugriff ab.

Funktionen

+run() : void	Ausführungsroutine des Threads.

10.7.8 Klasse PDKontroller (Singleton)

Der PDKontroller empfängt und verarbeitet die Nachrichten vom Netzwerk. Sie ist als Singleton realisert und ist die Schnittstelle vom Netzwerk zu der PD.

Attribute

-spielManager : SpielManager*	Pointer auf den Manager.
-serverFigur : Spielfigur*[]	Pointer auf die Spielfiguren (nur Server)
-clientFeld : SpielfeldClient*	Pointer auf das Spielfeld.

Funktionen

+setManager(SpielManager* manager) : void	Registriert den Spielmanager.
+setSpielfigur(int playerID, Spielfigur* figur) : void	Registriert eine Spielfigur.
+setClientFeld(SpielfeldClient* feld) : void	Registriert das Spielfeld.
Manager-Funktionen	
+newPlayer(int playerID, string playerName) : void	Ein neuer Spieler hat sich beim Server angemeldet.
+disconnect(int playerID) : void	Ein Spieler hat sich abgemeldet.
+disconnect() : void	Der Server hat sich abgemeldet.
+ready(int playerID) : void	Der Spieler ist spielbereit.
+run() : void	Spielstart
+endGame() : void	Beendet das Spiel.
+connectConfirm() : void	Bestätigt die Verbindung zum Server.
+receiveMsg(string msg) : void	Empfängt eine Nachricht vom Server.
+receiveMsg(int playerID, string msg) : void	Empfängt eine Nachricht vom Client (zum Verteilen an alle Clients).
+errorMsg(int msgNumber, string addInfo) : void	Empfängt eine Fehlermeldung vom Netzwerk.
+setPlayerName(int playerID, string name) :	Setzt den Namen eines Spielers.
Server-Funktionen	
+setDirection(int direction, int playerID) : void	Setzt neue Laufrichtung des Spielers.
+setBombPressed(bool keypressed, int playerID) : void	Setzt das Bomben-lege-Flag.
Client-Funtionen	
+setPlayfield(vector¡int¿ feldinfo, int zeilenbreite) : void	Initialisiert das ganze Spielfeld.
+setItem(Position position, int item) : void	Setzt ein einzelnes Objekt auf dem Spielfeld.
+setPlayer(int playerID, Position position) : void	Setzt die Position einer Spielfigur.
+delPlayer(int playerID) : void	Löscht einen Spieler vom Spielfeld.
+explode(Position position, int reichweite) : void	Löst eine Explosion auf dem Spielfeld aus.

10.7.9 Klasse Spielfeld

Die Klasse Spielfeld enthält ein Array das den aktuellen Zustand und die Positionen aller Spielelemente representiert. Sie hat Zugriff auf alle spielentscheidenden Informationen.

Attribute

-feld : Spielelement*[]	Abbild des aktuellen Spielstandes. Jeder Eintrag im Array entspricht einem Feld. d.h. es können nicht mehrere Elemente auf einem Feld sein. (Spielfiguren werden hier nicht gespeichert!)
-player : Spielfigur[]	Zeigerliste auf alle Spielfiguren des aktuellen Spiels. Die Position der Figur ist bei der Figur gespeichert.
-game : Spiel*	Pointer auf das Spiel-Objekt.
-netInterface : ServerInterface*	Pointer zum Netzwerk-Interface des Servers.
-minPlayers : int	Sind weniger Spieler als minPlayers auf dem Feld wird das Spiel beendet.

Funktionen

+startGame() : void	Startet das Spiel.
+stopGame() : void	Beendet das Spiel.
+setField(Position pos, int item, Spielfigur* figur = NULL) : void	Setzt ein bestimmtes Element an die Koordinate (x,y).
+getField(Position pos) : int item	Liefert das Element das sich an der Koordinate (x,y) befindet.
+setPlayer(int playerID, Spielfigur* player) : void	Meldet einen neuen Spielfigur beim Spielfeld an.
+delPlayer(int playerID) : void	Meldet den Spieler ab.
+explode(Position pos, int reichweite, queue¡Position¿* toDie = NULL) : void	Berechnet die Explosion.
-clearKilledElements(queue<Position>* toDie) : void	Löscht die gesprengten Elemente vom Spielfeld.

10.7.10 Klasse Spielfigur

Die Klasse Spielfigur enthält alle wichtigen Informationen über Position und Zustand der Spielfigur.

Attribute

-spielfeld : Spielfeld*	Pointer auf das Spielfeld-Objekt.
-pdKontroller : PDKontroller*	Pointer auf den den PD-Kontroller. Für ankommende Meldungen vom Netzwerk.
-netInterface : ServerInterface*	Pointer auf das Netzwerk-Interface. Für abgehende Meldungen ans Netzwerk.
-playerID : int	Spielernummer
-playerName : string	Name des Spielers.
-position : Position	Position der Spielfigur auf dem Spielfeld in (x,y) Koordinaten.
-direction : int	Richtung in die die Spielfigur gehen möchte. (STAY, UP, DOWN, LEFT, RIGHT)
-numberOfBombs : int	Die Anzahl Bomben die er noch legen darf. -1 wenn Bombe gelegt wurde, +1 wenn seine Bombe explodiert ist oder ein Bomben-Powerup aufgenommen wurde.
-reichweite : int	Reichweite der Bombe in Feldern. Wird der Bombe übergeben wenn sie gelegt wird. Wird erhöht, wenn ein Flammen-Powerup aufgenommen wird.
-alive : bool	TRUE wenn die Spielfigur noch lebt.
-bombPressed : bool	TRUE wenn die Bomben-lege-Taste gedrückt ist.
-moveTimer : QTimer*	Timer für die Bewegungs-Verzögerung.

Funktionen

+setDirection(int dir) : void	Setzt die Laufrichtung der Spielfigur (STAY, UP, DOWN, LEFT, RIGHT).
+setBombPressed(bool bomb) : void	Setzt das bombPressd Flag (bomben-lege-Taste gedrückt).
+getName() : string	Liefert den Namen des Spielers.
+getPosition() : Position	Liefert die aktuelle Position der Spielfigur.
+addBomb() : void	Fügt eine Bombe zum Arsenal der Spielfigur hinzu.
+wakeup() : void	Erweckt die Spielfigur zum Leben, erlaubt ihr sich zu bewegen.
+die() : void	Zerstört die Spielfigur wenn sie gesprengt wurde.
+move() : bool	Bewegt die Spielfigur um ein Feld in die aktuelle Richtung (direction).
+getReichweite() : int	Gibt die Reichweite zurück.
-dropBomb() : void	Legt eine Bombe an der aktuellen Position sofern noch Bomben im Arsenal.
-checkField(int posx, int posy) : int	Prüft ob ein Feld auf dem Spielfeld passierbar ist oder nicht.
-timerDone() : void	Wird aufgerufen wenn der moveTimer abgelaufen ist.

10.7.11 Klasse SpielerClient

Die Klasse Spieler empfängt die Benutzereingaben, bestimmt die daraus folgenden Aktionen und leitet sie an den Server weiter.

Attribute

-guiInterface : GUIToPDInterface*	Pointer zum GUI-Interface, zum Empfang der Benutzereingaben.
-netInterface : ClientInterface*	Pointer zum Netzwerk-Interface, zum Versenden der Aktionen.
-keylist : bool[5]	Speichert die Informationen welche Tasten momentan gedrückt sind.
-oldDirection : int	Die zuletzt übermittelte bewegungsrichtung.

Funktionen

+keyPressed(int key) : void	Eine Taste wurde gedrückt.
+keyReleased(int key) : void	Eine Taste wurde losgelassen.
-computeDirection() : void	Berechnet die Laufrichtung aus den Daten der keylist und sendet sie an den Server.

10.7.12 Klasse SpielfeldClient

Die Klasse SpielfeldClient hat ein vereinfachtes Abbild der Spielsituation gespeichert. Sie benötigt dieses zur berechnung der Explosionen. Sie empfängt alle Änderungen vom Server und gibt sie ans GUI weiter.

Attribute

-guiInterface : PDToGUIInterface*	Pointer zum GUI-Interface, zum Senden der Änderungen.
-pdKontroller : PDKontroller*	Pointer zum PD-Kontroller, zum Empfangen der Spielfeldänderungen.
-field : int[][]	Vereinfachtes Abbild der Spielsituation.
-playerPosition : Position[4]	Die Positionen der Spieler auf dem Spielfeld.
-readySet : bool	TRUE wenn bereit zum spielen.

Funktionen

+setPlayfield(vector¡int¿ feldinfo, int zeilenbreite) : void	Setzt das gesamte Spielfeld.
+setField(Position pos, int item) : void	Setzt ein bestimmtes Element an die Koordinate (x,y).
+setPlayer(int playerID, Position pos) : void	Setzt die Spielfigur (nr) an die Position (x,y).
+delPlayer(int playerID) : void	Löscht einen Spieler vom Spielfeld.
+explode(Position pos, int reichweite) : void	berechnet eine Explosion.

10.7.13 Klasse SpielElement

Oberklasse aller auf dem Spielfeld platzierbaren Elemente wie Mauer, Wand, Bombe und Powerup (ausser den Spielfiguren).

Attribute

-position : Position	Position des Elements.
-elementType : int	Typ des Elements.

Funktionen

+die() : SpielElement*	Rein Virtuelle Funktion die beim sprengen des Objekts aufgerufen wird.
+getType() : int	Liefert den Typ des Elements.

10.7.14 Klasse Mauer

Die Mauer ist vom Spielelement abgeleitet. Sie kann durch eine Explosion nicht zerstört werden. Sie hat keine spezielle Funktionalität.

10.7.15 Klasse Wand

Die Wand ist vom Spielelement abgeleitet. Sie kann durch eine Explosion zerstört werden und erzeugt dabei ev. ein Powerup.

Attribute

-powerupType : int	Typ des versteckten Powerups.

Funktionen

+getPowerupType() : int	Liefert den Typ des versteckten Powerups.

10.7.16 Klasse Bombe

Die Bombe ist vom Spielelement abgeleitet. Beim Erzeugen startet der Timer, der am Ende eine Explosion und damit die Zerstörung der Bombe einleitet. Sie kann durch eine Explosion zerstört werden und explodiert dabei selbst.

Attribute

-dropper : Spielfigur*	Pointer auf die Spielfigur die sie erzeugt hat.
-playfield : Spielfeld*	Pointer auf das Spielfeld auf dem sie liegt.
-explodeTimer : QTimer*	Timer zur Verzögerung der Explosion.
-reichweite : int	Die Reichweite der Explosion, wird beim Erzeugen gesetzt.

Funktionen

+getReichweite() : int	Liefert die Reichweite der Bombe.
-timeout() : void	wird vom Timer aufgerufen wenn er abgelaufen ist. Löst die Explosion aus.

10.8 Netzwerk Interface

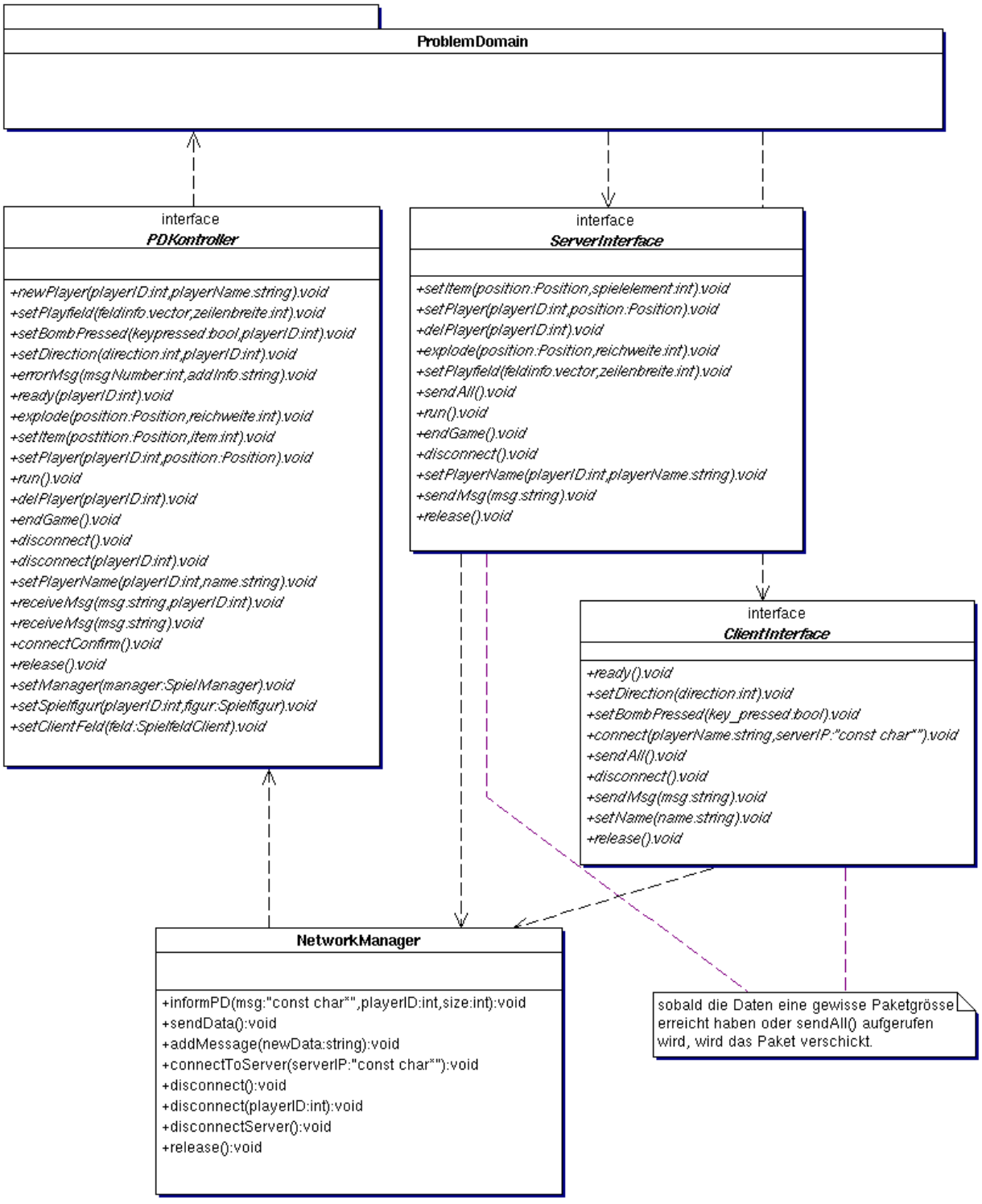

Abbildung 10.12: Interface zwischen Problem Domain und Netzwerk

Der Problem Domain stehen die als Singleton implementierten Klassen ServerInterface und Clientinterface zur Verfügung. Die Problem Domain eines Servers verwendet ausschliesslich das ServerInterface, um mit allen mitspielenden Clients zu kommunizieren.
Die Problem Domain eines Clients benutzt das ClientInterface, um sich bei dem Spielserver anzumelden und ihm Ereignisse zu melden.
Die Klassen ServerInterface und ClientInterface rufen ihrerseits Methoden der Klasse NetworkManager auf, welcher mit dem Netzwerk kommuniziert. Der NetworkManager wird von der Netzwerkschicht aufgerufen und ruft selbst Methoden aus der Klasse PDKontroller auf.
Die Schnittstelle generiert einen temporären String der verschiedene Protokollmessages enthält. Welche Protokollmessages eingepackt werden ist davon abhängig, welche Methoden des Serverinterfaces oder des Clientinterfaces aufgerufen wurden.
Dieses Paket wird abgeschickt, sobald es eine gewisse Länge erreicht hat, oder die Problem Domain dies durch einen Methodenaufruf auslöst.

10.9 Netzwerk Interface Klassenbeschreibung

Verwendete Konstante aus global.h zur Konfiguration des Verhaltens der Schnittstelle:
MAX_STRING_PAKET_SIZE. Übersteigt der temporär angelegte String der die Nutzdaten enthält diese Länge, wird er nach dem Einpacken der letzten Protokollmessage automatisch verschickt. Zum sofortigen Versenden der Daten kann sowohl im Serverinterface als auch im Clientinterface die Methode sendAll() aufgerufen werden. Als Singleton enthält diese Klasse auch eine Methode getServerInterface die einen Pointer auf die Schnittstelle zurückgibt.

10.9.1 Klasse ServerInterface

Diese Klasse wird von der Problem Domain des Spielservers benutzt, um Informationen an alle Spielclients zu schicken. Sie ist nach dem Singleton Pattern implementiert.

Funktionen

+setItem(Position _pos,int spielelement):void	ein einzelnes Spielelement wird platziert
+setPlayer(int playerID, Position _pos):void	die Spielfigur eines Spielers wird plaziert
+delPlayer(int playerID):void;	ein Spieler wird aus dem Spiel entfernt
+explode(Position _pos,int reichweite):void	signalisiert den Spielclients das Explodieren einer Bombe
+setPlayfield(vector<int> feldinfo,int zeilenbreite):void	verschickt Spielfelddaten; feldinfo enthält Spielelemente; zeilenbreite gibt die Anzahl Felder in einer Horizontalen des Spielfeldes an
+sendAll():void	löst das Versenden des zusammengesetzten Datenpaketes aus
+getServerInterface(): ServerInterface*	gibt eine Instanz des Serverinterfaces zurück
+run():void	Startet das Spiel auf den Clients
+endGame():void	Signalisiert den Clients das Spielende
+disconnect():void	Meldet den Server bei den Clients ab
+setPlayerName(int playerID,string playerName):void	Informiert die Clients über den Namen eines Spielers
+sendMsg(string msg):void	Verschickt an alle Clients eine Nachricht
+release():void	löscht den Singleton, wenn es keine Referenzen mehr darauf gibt

10.9.2 Klasse ClientInterface

Das Clientinterface wird von der Problem Domain des Spielclients benutzt, um Informationen an den Server zu schicken. Sie ist nach dem Singleton Pattern implementiert. Als Singleton enthält diese Klasse auch eine Methode getClientInterface die einen Pointer auf die Schnittstelle zurückgibt.

Funktionen

+ready():void	Spielclient meldet sich bereit
+setDirection(int direciton):void	Der Server wird über die Ausrichtung der Figur informiert
+setBombPressed(bool key_pressed):void;	Signalisiert, dass der Spieler Bomben legt
+connect(string playerName, const char* serverIP):void	Nimmt eine Verbindung zum Spielserver auf
+setName(string player_name):void	Setzt auf dem Server den Spielernamen
+sendAll():void	löst das Versenden des zusammengesetzten Datenpaketes aus
+getClientInterface(): ClientInterface*	gibt eine Instanz des Clientinterfaces zurück
+release():void	löscht den Singleton, wenn es keine Referenzen darauf mehr gibt
+disconnect():void	Meldet den Client beim Server ab
+sendMsg(string msg):void	fügt dem temporären Nachrichtenpaket(siehe 10.9.7) eine Nachricht(siehe 10.9.6) hinzu

10.9.3 Klasse NetworkManager

Der NetworkManager wird vom ClientInterface und dem ServerInterface verwendet, um mit dem Netzwerk zu interagieren. Er speichert die zu versendenden Nachrichten bis das Nachrichtenpaket eine gewisse Grösse hat (s.h. Nachrichtenpaket) oder dies von einem Interface verlangt wird.
Er erhält übers Netzwerk Nachrichtenpakete und wertet sie aus. Die erhaltene Information verwendet er um entsprechende Methoden im PDKontroller aufzurufen. Als Singleton enthält diese Klasse auch eine Methode getNetwrokManager die einen Pointer auf die Schnittstelle zurückgibt.

Funktionen

+getNetworkManager(): NetworkManager*	gibt eine Instanz des NetworkManagers zurück
+sendData():void	verschickt das zusammengesetzte Nachrichtenpaket(siehe 10.9.7) an den Server
+sendServerData():void	verschickt das zusammengesetzte Nachrichtenpaket(siehe 10.9.7) an alle Clients
+addMessage(string newData):void	fügt dem Nachrichtenpaket(siehe 10.9.7) eine Nachricht (siehe 10.9.6) hinzu
+informPD(const char* msg,int playerID,int size):void	interpretiert das Nachrichtenpaket(siehe 10.9.7) und ruft die entsprechenden Methoden in der Klasse PDKontroller auf
+connectToServer(const char* serverIP):void	stellt eine Verbindung zum Server her
+release():void	löscht den Singleton, wenn es keine Referenz mehr darauf gibt
+disconnect():void	Meldet den Client beim Server ab
+disconnect(int playerID):void	informiert die PD, falls ein Client nicht mehr erreicht werden kann
+disconnectServer():void	ist eine bereitgestellte Methode für das Abmelden des Servers

10.9.4 Klasse StringConverter

Der StringConverter ist eine Klasse die vom ServerInterface und dem ClientInterface benutzt wird um Nachrichtencodes und -daten in Pakete zu packen.
Der NetworkManager verwendet diese Klasse, um die Nachrichtenpakete wieder auszupacken und zu interpretieren. Dabei werden Zahlenwerte in Hexadezimalzahlen umgewandelt.

10.9.5 Netzwerkprotokoll

Die Kommunikation zwischen Server und Clients geschieht durch den Austausch von Nachrichtenpaketen , die verschiedene Nachrichten enthalten können. Jede Nachricht enthält zur Identifikation einen Nachrichtencode.

10.9.6 Nachricht

Eine Nachricht besteht aus einem Nachrichtencode(siehe 10.9.6) und einem Datenstring.

Nachrichtencodes

Die Nachrichtencodes charakterisieren den Typ einer Nachricht. Dieser Typ dient dem NetworkManager zur Identifikation der Methoden und dem Format der Nachrichtendaten. Er wertet die Nachricht aus und ruft die entsprechende Methode mit den entsprechenden Parametern im PDKontroller auf.

In der Datei global.h sind folgende Nachrichten Codes definiert:

SET_ITEM	= 0
SET_PLAYER	= 1
DEL_PLAYER	= 2
EXPLODE	= 3
SET_PLAYFIELD	= 4
REGISTER_PLAYER	= 5
READY	= 6
SET_DIRECTION	= 7
SET_BOMB_KEY	= 8
SET_NAME	= 9
END_GAME	= 10
RUN	= 11
DISCONNECT_SERVER	= 12
DISCONNECT_CLIENT	= 13

10.9.7 Nachrichtenpaket

Ein Nachrichtenpaket ist ein aus verschiedenen Nachrichten zusammengesetzter String. Eine einzelne Nachricht besteht aus einem Nachrichtencode und Daten. Es gibt aber auch Nachrichtentypen ohne zusätzliche Daten.

10.10 Netzwerk Klassendiagramm

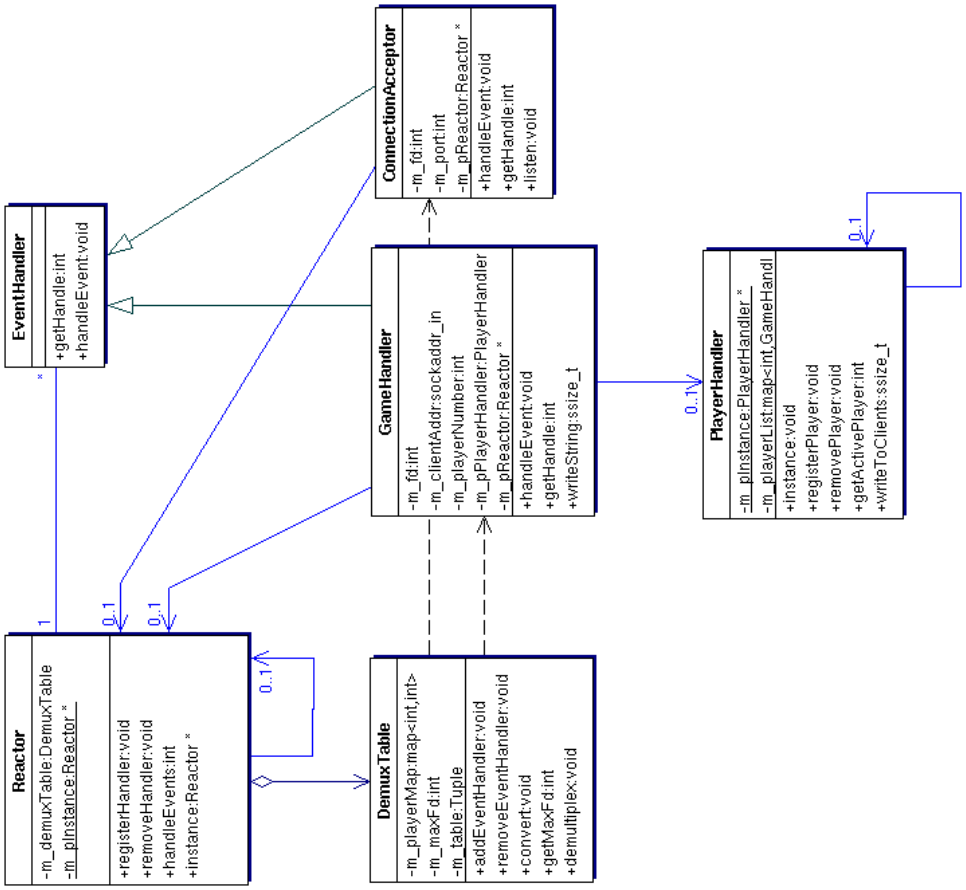

Abbildung 10.13: Klassendiagramm Netzwerk/Server

10.10.1 Beschreibung des Reactor Patterns

Das Problem, dass sich bei einem Spiel, das erstens gewissen Geschwindigkeitsanforderungen genügen muss und zweitens mehrere (mehr als 2) Mitspieler hat, ist, dass die Netzerwerkkommunikation speziell gelöst werden muss, da alle Clients zu einem nicht vorhersehbaren Zeitpunkt Meldungen zum Server schicken können. Da die Systemfunktionen, die für die Kommunikation benötigt werden (also read() und write()) blockierend sind, muss entweder für jeden Client ein seperater Prozess gestartet werden oder es muss ein geeignetes Pattern verwendet werden. Die erste Lösung wäre sehr einfach zu implementieren (wird auch sehr oft gemacht, zum Beispiel bei Web Anwendungen), hat aber den Nachteil, dass bei einer Anwendung wie der unsrigen Interprozesskommunikation erforderlich gewesen wäre, was sehr mühsam zu implementieren ist und auch Performance Nachteile mit sich bringt. Die zweite Lösung mit dem Pattern erschien uns deshalb vorteilhafter, obwohl es auch nicht einfacher zu implementieren ist, aber die sinnvollere Lösung für ein Spiel ist.

Das Reactor Pattern hat folgende Vorteile

- Wartezeiten und Antwortezeiten des Servers werden kürzer da nicht blockierend auf einen

Event eines einzigen Clients gewartet wird.

- Datendurchsatz wird erhört, da keine Daten zwischen einzelnen Prozessen ausgetauscht werden müssen.
- Sehr gute Wartungs- und Erweiterungseigenschaften, da Änderungen nur an einem Ort gemacht werden müssen.
- Es ist kein Multithreading und keine Synchronisation im Server nötig

Erreicht wird dies, indem synchron auf das Eintreffen von Ereignissen von verschiedenen Orten (Clients) gewartet wird. Diese Ereignisse werden entgegengenommen, ausgewertet und an die bereitgestellten services weitergeleitet. Die Klassen und Methoden des Reactor Patterns werden in den folgenden Abschnitten erklärt.

10.10.2 Klasse Reactor (Singleton)

Die Klasse ist dafür verantwortlich, die select() Funktion aufzurufen und anhand der Resultate, die sie von dieser Funktion erhält, Reaktionen auszuführen. Das heisst konkret, dass die select() Funktion, welche eine System Funktion ist, eine Meldung gibt, wenn ein Event eintrifft. Falls dies geschieht, ist die Reactor Klasse dafür verantwortlich, das richtige Handler Objekt (ConnectionAcceptor oder GameHandler) aufzurufen. Damit stellt diese Klasse eine Abstraktion der select() Funktion dar und ist dafür verantwortlich, dass die Handler Funktionen read() und connect() nur aufgerufen werden, wenn es tatsächlich nötig ist. Damit wird gewährleistet, dass das System nicht blockiert, sondern dass alle Clients sehr schnell bedient und abgefragt werden können.

Funktionen

+registerHandler(EventHandler* pEventHandler, EventType eventType) : void	registriert einen neuen Event Handler (ConnectionAcceptor oder GameHandler) mit dem dazugehörigen Event Typ in der Demultiplex Tabelle.
+removeHandler(EventHandler* pEventHandler, EventType eventType) : void	entfernt den Event Handler aus der Demultiplex Tabelle
+handleEvents() : int	führt die select() Funktion aus und ruft die entsprechende Methode im ConnectionAcceptor bzw. im GameHandler auf.
+instance() : Reactor*	gibt die Reactorinstanz zurück.

10.10.3 Klasse EventHandler

Diese Klasse ist eine rein virtuelle Klasse und stellt die Schnittstelle für die abgeleiteten Klassen ConnectionAcceptor und GameHandler dar.

10.10.4 Klasse ConnectionAcceptor

Der ConnectionAcceptor ist dafür zuständig, Verbindungsanfragen zu regeln. Das heisst, wenn sich ein Client mit dem Server verbinden möchte, macht der ConnectionAcceptor eine neue Verbindung auf der Serverseite (erstellt einen neuen Filedeskriptor) und falls es keine Fehler dabei gibt, wird ein neuer GameHandler erstellt.

Funktionen

+listen() : void	stellt einen Filedeskriptor mit der richtigen Struktur um Verbindungsanfragen entgegenzunehmen zur Verfügung und registriert diesen beim Reactor.
+handleEvent(int fd, EventType eventType) : void	ist eine überschriebene Funktion der Klasse EventHandler. Diese ist zuständig, ankommende Anfragen für eine Verbindung entgegenzunehmen und wenn kein Fehler auftritt, wird die Verbindung akzeptiert und ein neues GameHandler Objekt erstellt.
+getHandle() : int	gibt den Filedeskriptor, der in der listen() Funktion bereitgestellt wurde zurück.

10.10.5 Klasse GameHandler

Für jeden Client, der sich verbunden hat, gibt es ein GameHandler Objekt. Dieses Objekt regelt die ganze Kommunikation zwischen Client und Server für diesen bestimmten Client. Das heisst, er nimmt alles, was vom Client zum Server geschickt wird entgegen und schreibt alles vom Server zum Client.

Funktionen

+handleEvent(int fd, EventType eventType) : void	ist eine überschriebene Funktion der Klasse EventHandler. Sie nimmt die Strings, die vom Client an den Server übermittelt wurden an und gibt diese an die PD weiter.
+getHandle() : int	gibt den Filedeskriptor, der den GameHandler identifiziert zurück.
+writeToClient(const char* str, size_t n) : ssize_t	Schreibt die Strings, die von der PD an die Clients übermittelt werden sollen zu dem Client, für den das GameHandler Objekt zuständig ist.

10.10.6 Klasse DemuxTable

Die Demultiplex Tabelle ist dazu da, die Filedeskriptoren mit ihren entsprechenden Eventtypen zu registrieren, damit bei einer Anfrage des Clients das richtige GameHandler Objekt aufgerufen werden kann. Das heisst in dieser Tabelle sind alle File Deskriptoren mit den zugehörigen Events gespeichert.

Funktionen

+convert(fd_set& read_fds, fd_set& except_fds) : void	konvertiert die Event-Typen um herauszufinden, was für ein Event ansteht (ein read oder connect Event)
+addEventHandler(int fd, EventHandler* pEventHandler, EventType eventType) : void	ein EventHandler wird mit seinem Event Typ in der Tabelle registriert.
+removeEventHandler(int fd) : void	der EventHandler wird wieder aus der Tabelle entfernt.
+getMaxFd() : int	der Zahlenwert des grössten Filedeskriptors wird zurückgegeben.
+demultiplex(int fdCount, fd_set& read_fds, fd_set& except_fds) : void	es wird anhand des Event-Typs in der Tabelle und des anstehenden Events herausgefunden, welche handleEvent() Funktion aufgerufen werden muss.

10.11 Klasse PlayerHandler(Singleton)

Im Netzwerk werden noch Assoziationen zwischen Filedeskriptor und Player (also Client) gemacht. Die PlayerHandler Klasse ist von der Funktion her (nicht vom Aufbau) ähnlich wie die Demultiplex Tabelle. Sie speichert für jeden Client, also für jeden GameHandler einen Zeiger, damit sie diesen kennt. Wenn nun eine Meldung von der ServerPD an die Clients geschickt werden soll, wird in dieser Klasse die entsprechende write() Funktion in allen GameHandler Objekten aufgerufen.

Funktionen

+registerPlayer(GameHandler* pGameHandler) : void	Wenn sich ein neuer Spieler angemeldet hat, wird hier das zugehörige GameHandler Objekt, das den Spieler identifiziert, registriert
+removePlayer(GameHandler* pGameHandler) : void	Wenn die Verbindung zu einem Spieler nicht mehr besteht, wird der GameHandler dieses Spielers hier entfernt.
+getActivePlayer(GameHandler* pGameHandler) : int	gibt die Spielernummer (1-4) zurück.
+instance() : WriteHandler*	die Instanz des PlayerHandler wird zurückgegeben.

10.11.1 Client (Singleton)

Die Client Klasse steuert die ganze Verbindung auf der Seite des Clients. Er macht die Verbindung zum Server, schickt daten an diesen und liest auch die Daten, die vom Server geschickt werden.

Funktionen

+makeConnection(const char* strPtr): int	eröffnet eine Verbindung zum Server mit der angegebenen IP_Nummer und gibt den Filedeskriptor, der diese Verbindung identifiziert zurück.
+writeString(const char* str, size_t n) : ssizt_t	Schickt eine Meldung zum Server und gibt die Länge der geschriebenen Zeichenkette zurück
+readString() : ssize_t	liest die Meldung vom Server und gibt die Länge der Nachricht zurück. Die Funktion wird in einem eigenen Thread ausgeführt, damit eine Meldung möglichst schnell entgegengenommen werden kann.
+close() : void	schliesst die Verbindung zum Server
+instance() : WriteHandler*	die Instanz des Clients wird zurückgegeben.

10.12 Netzwerk Sequenzdiagramme

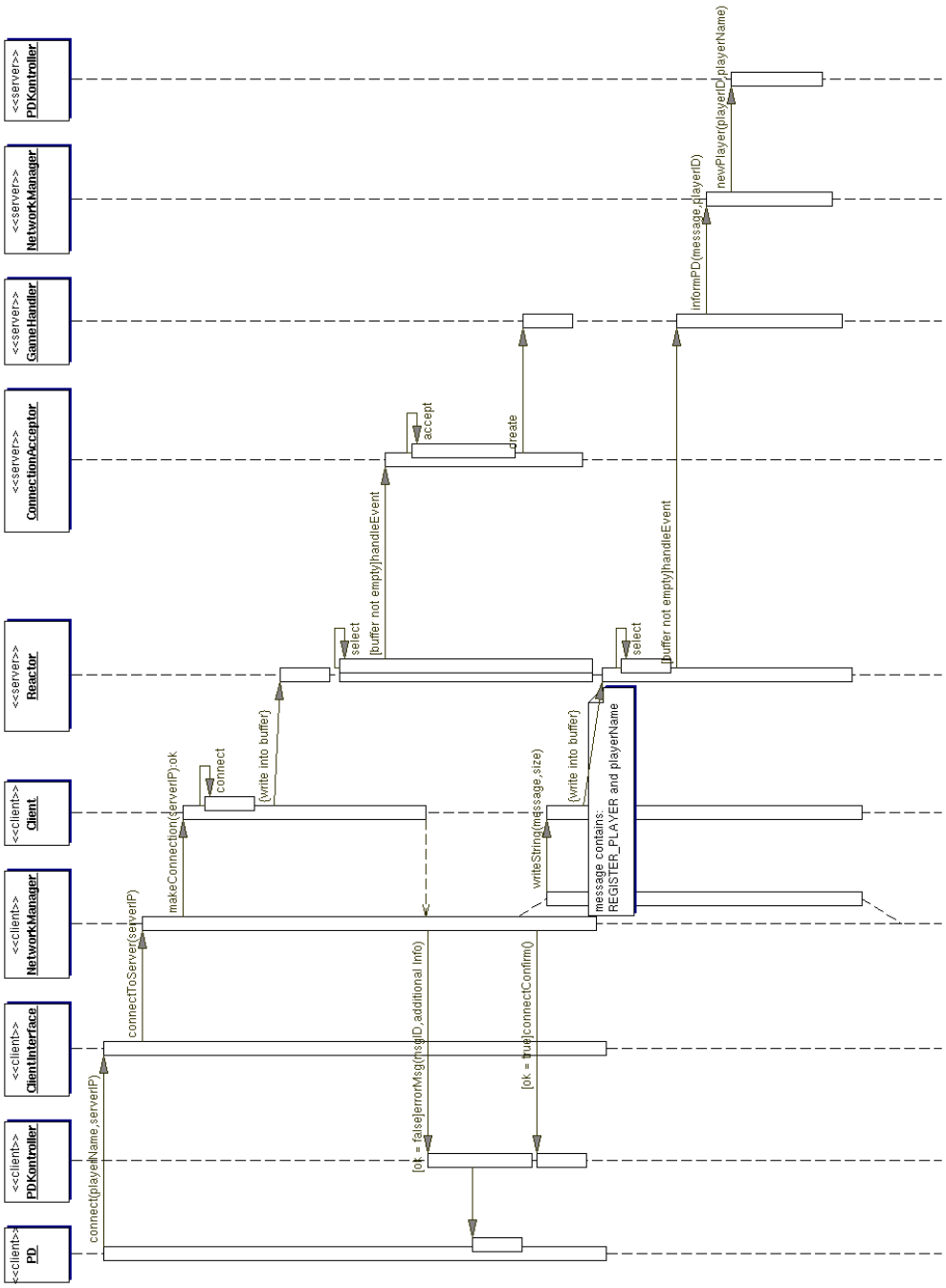

Abbildung 10.14: Anmeldung des Clients bei einem Spiel

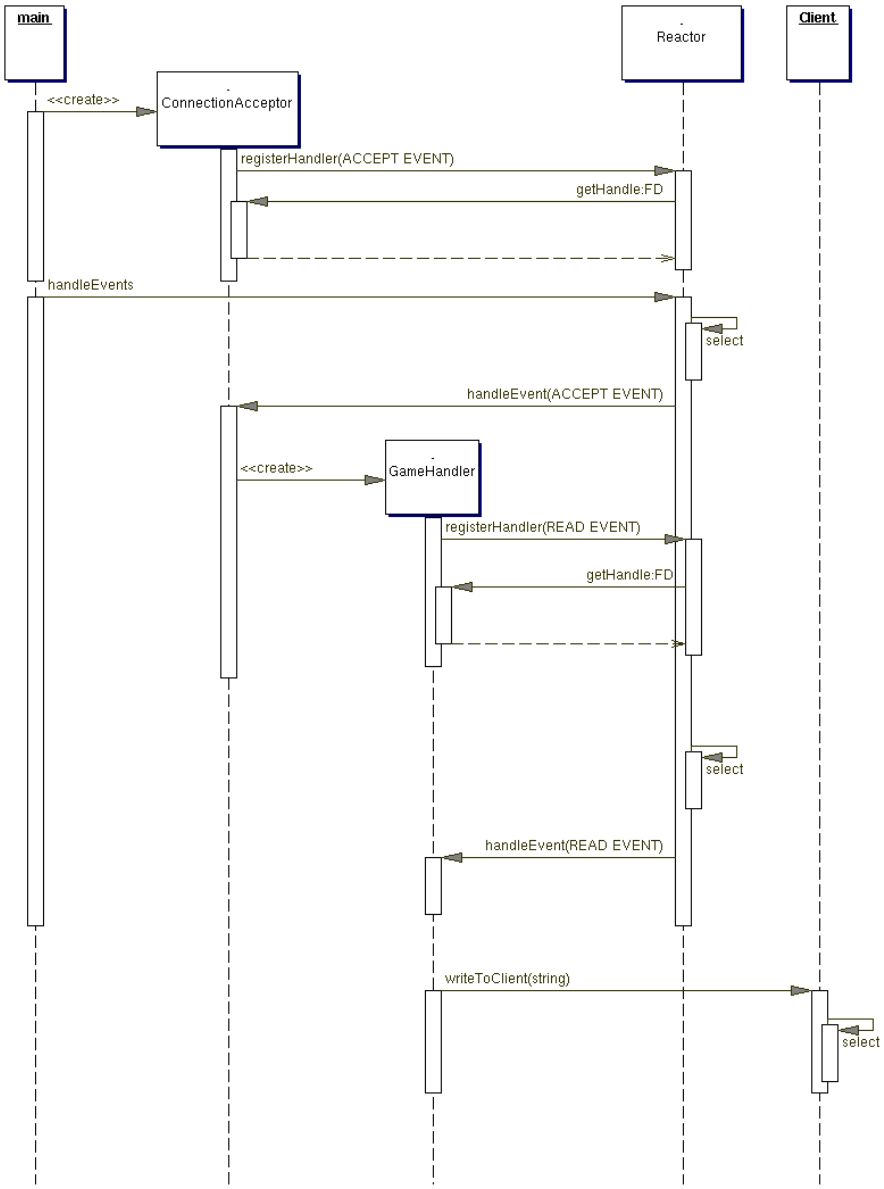

Abbildung 10.15: Sequenzdiagramm Ablauf Server

10.12.1 Erklärung Sequenzdiagramm Ablauf Server

Im main Programm wird zuerst ein Objekt des ConnectionAcceptor erstellt. Dieser registriert sich anschliessend beim Reactor und meldet damit, dass er für ACCEPT EVENTS, das heisst für ankommende Anfragen für eine Verbindung, zuständig ist. Der Reactor registriert den ConnectionAcceptor in der Demultiplex Tabelle. Damit ist der erste Schritt gemacht und der Server ist bereit, ankommende Verbindungsanfragen entgegenzunehmen.

Wenn das abgeschlossen ist, wird im Hauptprogramm (in einem eigenen Prozess) fortlaufend die Funktion handleEvents() des Reactors aufgerufen. Dieser führt den Systemaufruf select() durch, der prüft, ob eine Anfrage vorhanden ist. Ist dies der Fall, prüft er die Anfrage und im ersten Schritt wird das eine Verbindungsanfrage sein. Darauf ruft er handleEvent() im ConnectionAcceptor auf, der die Anfrage entgegennimmt und bearbeitet. Falls es dabei keine Fehler gibt, wird ein neuer GameHandler erzeugt, der sich gleich wieder beim Reactor registriert, diesmal aber für READ EVENTS. Danach ist der Ablauf derselbe wie im ConnectionAcceptor.

Von nun an wird fortlaufend geprüft, ob eine Anfrage anliegt und wenn ja, wird der Typ der Anfrage ermittelt (READ oder ACCEPT Event) und die enstprechende Funktion im richtigen Objekt aufgerufen.

Kapitel 11

Abschlusstest

Durchgeführt am: 11.7.2002
Tester : U. Heimann

11.1 Normaler Spielablauf

11.1.1 Spiel starten als Server

Aktion	**Erwartetes Ergebnis**	**Ergebnis**
Programm von der Konsole aus Starten.	Programm startet, GUI wird angezeigt.	OK
Spielername im Dialog 'Spieleinstellungen' setzen, mit 'Übernehmen' bestätigen.	Name wird übernommen (nicht sichtbar), Dialog verschwindet.	OK
Netzwerkserver starten, 'Netzwerk → Spielserver starten...'	Dialog 'Server starten' erscheint.	OK
Mit 'Starten' den Server starten.	Server wird gestartet (nicht sichtbar), der Spieler wird am eigenen Server mit dem eingegebenen Namen angemeldet. Es erscheint ein Dialog der die Verbindung bestätigt. Der eingegebene Name erscheint zuoberst in der Playerliste.	OK
Warten bis sich ein weiterer Spieler angemeldet hat.	Der Name des anderen Spielers erscheint in der Playerliste.	OK
Mit 'Spiel → Neues Spiel..' das Spiel starten.	Spielumgebung wird erstellt. Spielfelddaten werden übermittelt. Das Spielfeld wird angezeigt. Wenn sämtliche Spieler bereit sind wird das Spiel freigegeben.	OK

11.1.2 Spiel starten als Client

Aktion	Erwartetes Ergebnis	Ergebnis
Programm von der Konsole aus Starten.	Programm startet, GUI wird angezeigt.	OK
Spielername im Dialog 'Spieleinstellungen' setzen, mit 'Übernehmen' bestätigen.	Name wird übernommen (nicht sichtbar), Dialog verschwindet.	OK
An einem Server anmelden, 'Netzwerk → Spielserver anmelden...'	Dialog 'Server anmelden' erscheint.	OK
IP Adresse des Servers eingeben und mit 'Anmelden' bestätigen.	Der Spieler wird am Server mit dem eingegebenen Namen angemeldet. Es erscheint ein Dialog der die Verbindung bestätigt. Der eingegebene Name erscheint in der Playerliste.	OK
Warten bis sich weitere Spieler angemeldet haben.	Der Name der anderen Spieler erscheint in der Playerliste.	OK
Warten bis der Server das Spiel startet.	Spielumgebung wird erstellt. Spielfelddaten werden übermittelt. Das Spielfeld wird angezeigt. Wenn sämtliche Spieler bereit sind wird das Spiel freigegeben.	OK

11.1.3 Spielen

Aktion	Erwartetes Ergebnis	Ergebnis
Mit den Pfeiltasten wird die Spielfigur auf den freien Feldern bewegt.	Die eigene Spielfigur bewegt sich gemäss den Anweisungen.	OK
Mit der Leertaste wird eine Bombe gelegt.	Auf dem Spielfeld an der Position der Spielfigur wird eine Bombe angezeigt.	OK
Die Bombe explodiert nach einer bestimmten Zeit von selbst.	Auf dem Spielfeld wird der Feuerstrahl angezeigt. Wände die vom Feuerstrahl getroffen werden, werden gelöscht. Der Feuerstrahl verschwindet von selbst wieder.	OK
Mehrere Bomben werden zeitlich versetzt nebeneinander gelegt.	Die Explosion der ersten Bombe löst die anderen Bomben ebenfalls aus.	OK
Der andere Spieler bewegt seine Spielfigur und legt Bomben.	Die Spielfigur des Gegners bewegt sich auf dem Spielfeld gemäss den Anweisungen des Gegners auf dem anderen Computer. Bomben werden angezeigt und explodieren.	OK
Eine Spielfigur wird von einem Feuerstrahl getroffen.	Die getroffene Spielfigur wird vom Spielfeld entfernt. Wenn nur noch ein Spieler auf dem Feld ist, erscheint eine Meldung mit dem Namen des Siegers. Das Spiel wird beendet.	OK
Der Server startet mit 'Spiel → Neues Spiel..' ein neues Spiel.	Ein neues Spielfeld wird angezeiget. Alle Spieler sind wieder dabei und können mitspielen.	OK

11.1.4 Spiel beenden

Aktion	Erwartetes Ergebnis	Ergebnis
Ein Spieler (nicht der Server) schliesst seine Anwendung.	Der Spieler wird beim Server abgemeldet und das Programm wird geschlossen. Die verbliebenen Spieler erhalten eine Meldung, dass sich der Spieler abgemeldet hat. Der Name des Spielers verschwindet aus der Playerliste.	OK
Der Spieler mit dem Server schliesst seine Anwendung.	Der Server meldet sich bei allen Clients ab und das Programm wird geschlossen. Die verbliebenen Spieler erhalten eine Meldung, dass sich der Server abgemeldet hat. Sämtliche Namen der Spieler verschwinden aus der Playerlist.	OK

11.2 Varianten Spielablauf

11.2.1 Spiel starten

Aktion	Erwartetes Ergebnis	Ergebnis
Spiel vom Konqueror aus starten.	Spiel startet, GUI wird angezeigt.	FAILED Das Spiel wird gestartet, die Grafiken für das Spielfeld werden jedoch nicht gefunden.
Ein neuer Spieler versucht sich während eines laufenden Spiels eine Verbindung aufzubauen.	Der Spieler kann die Verbindung aufbauen, kann aber keinen Einfluss auf das laufende Spiel nehmen. Beim nächsten Spielstart spielt der Spieler auch mit.	OK
Ein Spieler gibt vor dem Anmelden am Server keinen Spielernamen ein.	Er erscheint in der Playerliste als 'anonymous'.	OK

11.3 Fehlerfälle

Aktion	Erwartetes Ergebnis	Ergebnis
Der Spieler gibt beim Verbinden mit einem Server eine ungültige IP ein.	Es wird eine Meldung ausgegeben, dass mit dem Server keine Verbindung aufgebaut werden konnte.	OK
Mitten im Spiel schliesst ein Spieler (nicht der Server) seine Anwendung.	Die Figur des Spielers wird zerstört und verschwindet vom Spielfeld. Der Name des Spielers wird aus der Playerliste entfernt. Das Spiel läuft weiter.	OK
Mitten im Spiel schliesst der Server seine Anwendung.	Das Spiel wird beendet. Die verbliebenen Spieler erhalten eine Meldung, und sämtliche Namen werden aus der Playerliste gelöscht.	OK

Kapitel 12

Buglist

Nr.	Fehlerbild	Mögliche Ursache	Status
1	Spielfiguren werden erst angezeigt wenn sie bewegt werden.	Positionen der Figuren werden beim Spielfeldaufbau nicht übermittelt.	FIXED
2	Wenn ein Spieler die Verbindung abbricht wird der Name in der Playerliste nicht gelöscht.	Im GUI wird auf das 'removePlayer' Ereigniss nicht reagiert.	FIXED
3	Wenn das Programm beendet wird muss ca. 30 - 60 Sekunden gewartet werden bis wieder ein Server gestartet werden kann.	Eventuell wird der verwendete Socket vom System nicht unmittelbar freigegeben.	
4	Wenn das Spiel vom Konqueror aus gestartet wird, wird das Spielfeld nicht angezeigt. Das Spiel funktioniert sonst normal.	Die Grafiken der Wände, Mauern, etc. können nicht geladen werde. Eventuell falscher Arbeitspfad.	

Kapitel 13

Installation und Bedienung

13.1 Installation

Die Installation erfolt wie bei den meisten Programmen unter Linux. Zuerst muss das tar Archiv (falls nicht schon entpackt) mit dem Befehl

```
tar -xvzf <name>.tgz
```

entpackt werden. Anschliessend wechseln Sie in das neu erstellte Verzeichnis und kompilieren und installieren das Programm. Das geschieht mit den folgenden Befehlen

```
cd netbomber
make
make install
```

Eine Ausführlichere Beschreibung finden sie in der mitgelieferten README Datei.

13.2 Bedienungsanleitung

Das Spiel NetBomb können Sie mit dem von der Konsole aus starten. Dazu wechseln Sie zuerst in das Verzeichnis, in das sie NetBomb installiert haben. Anschliessen geben Sie die Befehle

```
cd netbomber
./netbomber
```

ein. Damit wird das Programm gestartet. Es erscheint ein Fenster mit dem Spiel.

Als erstes können Sie unter dem Menu *Optionen* → Tastaturbelegung Ihren Namen eingeben. Übernehmen Sie Ihre Eingabe mit der Taste *Übernehmen* und Beenden Sie den Dialog mit *Abbrechen*. Anschliessend haben Sie zwei Möglichkeiten. Sie können

1. Einen Server starten, bei dem sich ihre Mitspieler anmelden können. Dazu gehen Sie folgendermassen vor
   (a) Wählen Sie *Netzwerk* → *Spielserver starten* und wählen Sie dann die Anzahl Mitspieler. Anschliessen können Sie mit der Taste *Starten* den Server starten.
   (b) Es erscheint eine Meldung die Ihnen Mitteilt, ob sie erfolgreich eine Spielserver starten konnten. Bestätigen sie diesen Dialog. Danach erscheint in der Spielerliste (Liste rechts oben) ihr Name.
   (c) Wenn Sie erfolgreich einen Server gestartet haben, müssen Sie warten, bis sich Ihre Mitspieler auf Ihren Server eingeloggt haben. Sie sehen die Spieler, die Spielbereit sind in der Spielerliste.

(d) Sobald genügend Spieler angemeldet sind, können sie mit *Spiel → Neues Spiel* ein neues Spiel starten.

(e) Have fun!

2. Sich bei einem Server anmelden, um mitzuspielen. Dazu gehen Sie folgendermassen vor

   (a) Sie können sich mit *Netzwerk → Spielserver anmelden* bei einem Server anmelden.

   (b) Es erscheint ein Fenster, in den Sie die IP - Adresse des Servers eingeben müssen. Tun Sie das und bestätigen Sie die Eingabe mit *Anmelden* oder brechen Sie die Eingabe *Abbrechen* ab.

   (c) Sobald der Spieler, der den Server gestartet hat, beginnt das Spiel

   (d) Have fun!

Sie können das Spiel jederzeit mit *Spiel → Beenden* schliessen.

Viel Spass beim Spielen. Das NetBomb - Team

Anhang A

Glossar

A.1 Namen

STEK	Stefan Künzle
REH	René Herrmann
MIE	Michael Egli
UHEI	Urs Heimann

A.2 Spielbegriffe

Spielfeld	Besteht aus freien Feldern, Mauern und Wänden. Der Spieler kann seine Figur auf den freien Feldern bewegen.
Freies Feld	Darauf kann sich die Spielfigur frei bewegen und Bomben legen. Es können sich Powerups darauf befinden.
Mauer	Ein dauerhaftes Hinderniss auf dem Spielfeld. Sie kann nicht durch eine Bombe zerstört werden und bleibt während dem ganzen Spiel unverändert.
Wand	Ein Hindernis auf dem Spielfeld das durch eine Bombe zerstört werden kann. Sie wird dann zu einem freiem Feld. Sie kann ein Powerup enthalten das nach der Sprengung auf dem Feld liegenbleibt.
Spielfigur	Wird vom Spieler auf den freien Feldern bewegt und kann Bomben legen. Sie kann nicht durch Mauern oder Wände gehen. Tritt sie auf ein Feld mit einem Powerup nimmt sie dieses auf. Wird sie von einem Feuerstrahl getroffen, stirbt sie und verschwindet vom Spielfeld.
Bombe	Wird von der Spielfigur auf das Spielfeld gelegt und explodiert nach einer bestimmten Zeit. Beim explodieren erzeugt sie für jede Himmelsrichtung einen Feuerstrahl.
Feuerstrahl	Auswirkung der explodierenden Bombe. Er ist ein oder mehrere Felder lang und zerstört Wände, Spielfiguren und Powerups. Seine Länge hängt davon ab von welcher Spielfigur die Bombe gelegt wurde und wieviele Flamme-Powerups diese aufgenommen hat.
Serie	Mehrere hintereinander gelegte Bomben bevor die erste explodiert. (zu beginn nur eine Bombe in Serie möglich)
Bombe-Powerup	Liegt auf einem freien Feld und kann von der Spielfigur aufgenommen werden. Es ermöglicht ihr eine Bombe mehr in Serie zu legen.
Flamme-Powerup	Kann von der Spielfigur aufgenommen werden und erhöht die Reichweite seiner Bomben um ein Feld.
Spielelement	Sammelbegriff für Mauer, Wand, Bombe, Powerups.
Ojekt des Spielfeldes	Spielelemente, Spielfigur

A.3 Technische Begriffe

GUI	Grafisches User Interface. Über dieses interagiert der Benutzer, in unserem Fall der Spieler.

Anhang B

Änderungsgeschichte

Datum	**Wer**	**Version**	**Kapitel**	**Änderung**
06.04.02	alle	0.1	1-3	Erstellung Kapitel 1-3 (MS 1)
06.04.02	REH,MIE	0.1.1	1-3	Überarbeiten V 0.1
12.04.02	alle	0.2	4	Erstellung Kapitel 4 (MS 2)
19.04.02	alle	0.2.1	4,5	Vervollständigt und korrigiert
26.04.02	alle	0.3	5,6	Kap 5 überarbeitet und Kap. 6 neu erstellt
07.05.02	STEK,UHEI	0.4	6	GUI und PD Design eingefügt
12.06.02	alle	1.0	alles	überarbeiten und vervollständigen zur Abgabe