# Informatikprakikum CE2
# Übung 2: Adventure - Spiel

Torben Nehmer
*Torben.Nehmer@gmx.net*

Wolfgang Nitz
*nitz@alpha.fh-furtwangen.de*

Peter Allgeier
*P.Allgeier@vega-g.de*

Nina Siraky
*siraky@foo.fh-furtwangen.de*

Thomas Graf
*ThomasGraf@swol.de*

Erstellt am 31. August 1999

# Inhaltsverzeichnis

# 1 Vorwort

Anhand der Programmierung eines Text - Adventures lassen sich sehr einfach die Prinzipien der Objektorientierung demonstrieren, da man sich dort den Begriff “Objekt” sehr leicht verständlich machen kann. Es existiert der Bezug zur Realität. Dies beginnt bei Gegenständen, die aufgehoben werden und endet beim Spieler selbst. All diese Elemente lassen sich jeweils in ein Objekt kapseln. Dabei sind besonders zwei Objektgruppen interessant: Gegenstände und Räume. Sie können alle vom gleichen Basisobjekt abgeleitet werden und stellen so eine ideale Umgebung zur Einführung der Vererbungsmechanismen dar.

Da diese Übung lediglich als Grundstein für das eigentliche Textadventure dient, besitzen die einzelnen Klassen bisher keine speziellen Funktionen. Gegenstände können aufgehoben und abgelegt werden, Räume können Gegenstände und Verbindungen zu anderen Räumen enthalten, der Spieler kann Gegenstände aufnehmen oder ablegen und sich innerhalb des Labyrinths bewegen. Weitere Funktionalität ist nicht vorhanden.

**Bezugsquellen der Dateien:** Alle Dateien, die in dieser Ausarbeitung vorgestellt wurden, sind im Internet unter *http://asterix.ghb.fh-furtwangen.de/studium/index.html* erhältlich.

Die Ausarbeitung und den Quellcode der einfach vorwärtsverketteten Liste, die in den vorgestellten Klassen verwendet wird, sind dort ebenfalls zu finden.

# 2 Beschreibung der Klassenstruktur

Das Rollenspiel läßt sich durch vier Klassen darstellen:

- `CMeister` stellt die zentrale Klasse zur Spielkonfiguration dar. Mit ihr wird das Labyrinth, die Gegenstände und die verschiedenen, möglichen Zuordnungen erstellt.
- `CSpieler` wird von `CMeister` kreiert und übernimmt zum eigentlichen Spielbeginn die Kontrolle. Dies beinhaltet momentan die Navigation durch das Labyrinth und das Aufnehmen oder Ablegen von Gegenständen.
- `CRaum` repräsentiert ein Raum des Labyrinths. Echte funktionalität besitzen diese Klasse derzeit noch nicht. Es werden lediglich Gegenstände und Wege zu anderen Räumen gesichert.
- `CGegenstand` entspricht den einzelnen Objekten, die entweder im Labyrinth liegen oder sich in den Taschen des Spielers befinden. Dies ist derzeit lediglich ein minimals Datenobjekt, daß später als Abstrakte Basisklasse verwendet werden sollte.

Alle Klassen, die Bildschirmeingaben verarbeiten sollen, erhalten Nachrichtenbehandlungsfunktionen. Diese sind privat und werden nur von Ereignissen ausgelöst, die durch die definierten Schnittstellenfunktionen ausgelöst werden.

## 2.1 “CMeister” - Der Meister der Labyrinthe

Die Klasse `CMeister` übernimmt direkt nach dem Programmstart die Kontrolle über das Spiel. Mit ihrer Hilfe kann der Benutzer das Labyrinth erstellen. Ist dies geschehen, so wird von ihm das eigentliche Spiel gestartet, und die Kontrolle geht an den Spieler über. Eine Sinnvolle Erweiterung währe die Möglichkeit, aktuelle Zustände speichern und laden zu können, da derzeit das Labyrinth bei jedem Programmstart neu erzeugt werden muß.

Der Meister koordiniert somit den Labyrinthaufbau und stellt die Datenintegrität vor dem Spielbeginn sicher, damit Prüfungen solcher Art innerhalb des Spielers nicht notwendig werden.

### 2.1.1 Konstruktion / Destruktion

```
class CMeister
{...
public:
    CMeister (string strName);
    virtual ~CMeister();
...};
```

Der Konstruktor intialisiert alle Datenstrukuten auf gültige Werte, indem er die notwendigen Objekte erstellt. Da alle Speicheranforderungen sich in einer Änderung in einer der beiden Listen `m_plistRaum` bzw. `m_plistGegenstand` niederschlägt wird dies vom Destruktor genutzt um alle Speicherbereiche die angefordert wurden auch wieder freizugeben.

### 2.1.2 Datenstrukturen

**Speicherverwaltung:**

```
class CMeister
{...
private:
    CList<CRaum *> * m_plistRaum;
    CList<CGegenstand *> * m_plistGegenstand;
    CSpieler * m_pSpieler;
    string m_pstrName;
...};
```

Da `CMeister` die zentrale Labyrinthverwaltung darstellt, ist sie für das vollständige Freigeben des angeforderten Speicherraums für die einzelnen Räume und Gegenstände verantwortlich. Zu diesem Zweck existieren zwei Listen, in denen alle vorhandenen Räume und Gegenstände gespeichert werden. Diese Listen werden auch von den Labyrinth - Konfigurationsroutinen verwendet, um Listen aller Räume oder Gegenstände anzuzeigen.

`m_pSpieler` wird die Adresse des aktuellen Spielerobjektes gespeichert, `m_pstrName` enthält den Namen des aktuellen Spieles.

**Spielzustand:**

```
class CMeister
{...
private:
    CRaum * m_praumBeginn;
    CRaum * m_praumEnde;
    CList<CGegenstand *> * m_plistInventar;
...};
```

Die beiden `CRaum` Zeiger definieren den Spielstart und das zu suchende Ende.

Die Liste `m_plistInventar` wird beim Spielstart dem Spieler übergeben. Sie stellt das aktuelle Inventar des Spielers dar. Wird das Spiel von diesem beendet, so übergiebt er das aktualisierte Inventar zurück an `CMeister`.

### 2.1.3 Funktionen des Nachrichten Handlers

```
class CMeister
{...
private:
  // Gegenstandsverwaltung
    virtual int msgGegenstandErstellen();
    virtual int msgGegenstandZuordnen();
    virtual int msgGegenstandLoeschen();
  // Raumverwaltung
    virtual int msgRaumErstellen();
    virtual int msgRaumZurodnen();
    virtual int msgRaumLoeschen();
  // Listenausgabe
    virtual int msgGegenstandListe();
    virtual int msgRaumListe();
  // Spielerverwaltung
    virtual int msgSpielerAnlegen();
    virutal int msgSpielerLoeschen();
  // Spielzustand
    virutal int msgStartEnde();
    virtual int msgSpielstart();
...};
```

Diese Hilfsfunktionen dienen zur Nachrichtenbehandlung. Sie werden von der Nachrichtenschleife `CMeister::Start()` aufgerufen. Die virtuelle Deklaration ermöglicht das anpassen auf eine erweiterte Spielstruktur. Sie übernehmen beim Aufruf jeweils die Kontrolle über Bildschirm und Tastatur.

Diese Menüstruktur besitzt einige Schwachpunkte, die in der dritten Übung beseitigt werden. Derzeit besteht keine Möglichkeit, einmal zugeordnete Gegenstände zu einem anderen Objekt zuzuordnen. Dazu muß der Gegenstand zuerst gelöscht und dann neu erstellt werden. Auch sind Funktionen zum Speichern und Laden des Labyrinthaufbaus und des eigentlichen Spielzustandes notwendig.

**Gegenstandsverwaltung:** Mit Hilfe dieser Funktionen lassen sich die Gegenstände verwalten. Dabei ist eine Zuordnung sowohl an den Spieler als auch an einen Raum möglich. Auf Grund der Aufgabenstellung sind hier noch keine komplexeren Funktionen wie Ändern einer Zordnung implementeiert.

**Raumverwaltung:** Analog zur Gegenstandverwaltung lassen sich hier Räume erstellen und löschen. Die Raumzuordnung ermöglicht es, Verbindungen zwischen den einzelnen Räumen aufzubauen.

**Listenausgabe:** Dies sind Hilfsfunktionen, die sowohl direkt durch den Benutzer, als auch indirekt durch andere Nachrichten Handler aufgerufen werden. Sie geben eine komplette Liste der Gegenstände bzw. der Räume aus.

**Spielerverwaltung:** Diese Menüpunkte erstellen und löschen den aktuellen Spieler.

**Spielzustand:** Der Nachrichten Handler `msgStartEnde` legt Start- und Zielraum fest. Sind diese Informationen vorhanden, kann das Spiel mit `msgSpielstart` gestartet werden. In diesem Moment geht die Kontrolle an die Klasse `CSpieler` über.

### 2.1.4 Öffentliche Zugriffsfunktionen

```
class CMeister
{...
public:
    virtual string GetName ();
    virtual int CheckEnde (CRaum * pRaum);
    virtual int Start ();
...};
```

Außer dem Konstruktor und dem Destruktor sind dies die einzigen öffentlichen Memberfunktionen der Klasse `CMeister`. Die Funktion `CMeister::GetName(...)` liefert den Namen des aktuellen Spieles. Die Menüschleife wird durch die Funktion `CMeister::Start()` eingeleitet. `CMeister::CheckEnde(...)` wird derzeit von der Klasse `CSpieler` benötigt, um festzustellen, ob das Spielziel erreicht ist.

## 2.2 “CSpieler” - Der Leidtragende

Ist der Aufbau des Labyrinthes mit Hilfe der Klasse `CMeister` abgeschlossen, so kann das Spiel beginnen: `CSpieler` übernimmt die Kontrolle.

### 2.2.1 Konstruktion / Destruktion

```
class CSpieler
{...
public:
    CSpieler (string strName, CMeister * pMeister);
    virtual ~CSpieler();
...};
```

Der Konstruktor initialisiert die dem Spieler eigenen Hilfsvariabeln. Da `CSpieler` keine neuen Objekte erstellt, sind hier keine erweiterten Prüfungen auf angeforderten Speicherplatz notwendig.

### 2.2.2 Datenstrukturen

```
class CSpieler
{...
private:
    CRaum * m_praumAkt;
    CList<CGegenstand *> * m_plistGegenstand;
    string * m_pstrName;
    CMeister * m_pMeister;
...};
```

Diese Datenstrukturen repäresentieren den aktuellen Zustand des Spielers: Seine Position, sein Inventar, sein Name und die Verbindung zur zugeordneten `CMeister`-Klasse.

### 2.2.3 Private Hilfsfunktionen

```
class CSpieler
{...
private:
    int TakeGegenstand (int iPos);
    int DropGegenstand (int iPos);
    int LeaveRaum (int iPos);
...};
```

Diese Funktionen stellen eine einfache Möglichkeit dar, Gegenstände aufzuheben oder abzulegen und in einen anderen Raum zu wechseln. Sie sind als Hilfsfunktionen primär als Arbeitserleicherung gedacht. Der Index `iPos` gibt jeweils die Position des zu bearbeitenden Objekts an. Es wird also beispielsweise mit `pSpieler->LeaveRaum(5);` in den fünften angrenzenden Raum gewechselt.

### 2.2.4 Funktionen des Nachrichten Handlers

```
class CSpieler
{...
private:
  // Listenausgaben
    virtual int msgShowWege();
    virtual int msgShowGegenstaende();
    virtual int msgShowInventar();
  // Kommandos
    virtual int msgMove();
    virtual int msgLook();
    virtual int msgTakeGegenstand();
    virtual int msgDropGegenstand();
...};
```

Wie in `CMeister` existiert eine Nachrichtenbehandlungsschleife, die verschiedene virtuelle Menüfunktionen aufruft.

**Listenausgaben** Diese Hilfsfunktionen werden auch hier sowohl direkt vom Benutzer als auch indirekt über verschiedene Kommandos ausgelöst.

**Kommandos** Sie stellen das eigentliche Spiel dar, in dem sie Navigation und Interaktion mit dem Labyrinth ermöglichen.

### 2.2.5 Öffentliche Zugriffsfunktionen

```
class CSpieler
{...
public:
    virtual string GetName ();
    virtual int Start (CRaum * pStart,
                       CList<CGegenstand *> * plistInventar);
...};
```

Die Startfunktion aktiviert die Menüschleife. bevor sie jedoch in diese Eintritt setzt sie die Spielerposition auf `pStart` und kopiert das übergebene Inventar `plistInventar` in die eigenen Membervariabeln. Beendet der Benutzer das Spiel, so wird das aktualisierte Spielerinventar wieder über `plistInventar`

an `CMeister` zurückgegeben. Dadurch ist es möglich, das Spiel kurzzeitig zu verlassen um einige Modifikationen am Labyrinth vorzunehmen und danach weiterzuspielen.

Die Funktion `CSpieler::GetName(...)` liefert den Namen des aktuellen Spielers zurück.

## 2.3 “CRaum” - Wo bin ich?

`CRaum` ist derzeit eine einfache Datenklasse, die außer Zugriffsfunktionen auf die im Raum enthaltenen Gegenstände und die dem Raum zugeordneten Wege derzeit keine echte Funktionalität bietet. Dabei werden lediglich die Funktionen der Listenklasse gekapselt. Für Übung 3 eigent sich diese Raumklasse sehr gut als Basisklasse, da sie sehr einfach mit spezifischer Funktionalität erweitert werden kann.

### 2.3.1 Konstruktion / Destruktion

```
class CRaum
{...
public:
    CRaum (string strName);
    virtual ~CRaum ();
...};
```

Konstruktor und Destruktor initialisieren bzw. zerstören die vorhandenen Datenstrukturen.

### 2.3.2 Datenstrukturen

```
class CRaum
{...
private:
    CList <CRaum *> * m_plistRaum;
    CList <CGegenstand *> * m_plistGegenstand;
    string * m_pstrName;
...};
```

Mit Hilfe dieser beiden Listen wird die interne Struktur des Labyrinths erstellt. Verbindungen zwischen einzelnen Räumen und plazierte Gegenstände schlagen sich hier nieder.

`m_pstrName` enthält den Namen des aktuellen Raumes.

### 2.3.3 Öffentliche Zugriffsfunktionen

```
class CRaum
{...
public:
    virtual string GetName ();
  // Zugriffsfunktionen für angrenzende Räume
    int AddRaum (CRaum * pRaum);
    int GetRaum (int iPos, CRaum * & pRaum);
    int RemoveRaum (int iPos, CRaum * & pRaum);
  // Zugriffsfunktionen für im Raum befindliche Gegenstände
    int AddGegenstand (CGegenstand * pGegenstand);
    int GetGegenstand (int iPos, CGegenstand * & pGegenstand);
    int RemoveGegenstand (int iPos, CGegenstand * & pGegenstand);
...};
```

Die Zugriffsfunktionen sind jeweils direkt auf die der Liste abgebildet. Die *Add* - Funktionen hängen einen Wert an die Liste an, die *Get* - Funktionen lesen einen Wert aus der Liste und die *Remove* - Funktionen entfernen einen Wert aus der Liste und geben ihn zurück.

Eine zugriffsfunktion auf den Namen des Raumes steht ebenfalls zur Verfügung.

## 2.4 “CGegenstand” - Wo hatt’ ich das noch gleich?

Bei `CGegenstand` handelt es sich derzeit um eine Dummy-Klasse. Ausser ihren Namen aufzuzeichnen besitzt sie keine Funktionalität. Im endgültigen Textadventure wird sie als reine, abstrakte Basisklasse auftreten.

```
class CGegenstand
{
private:
    string m_pstrName;

public:
    CGegenstand (string strName);
    virtual ~CGegenstand ();

    virtual string GetName ();
};
```

# 3 Schnittstellenbeschreibung der einzelnen Klassen

Alle Funktionen liefern einen Fehlercode als Rückgabewert. Ist dieser int gleich `0`, so ist kein Fehler aufgetreten. Verschieden Werte ungleich `0` zeigen mögliche Fehlerzustände an.

Alle Klassen haben eine gemeinsame Schnittstellenfunktion, die den Namen des Objektes in Form eines String - Objektes zurückgiebt:

```
virtual string ...::GetName ();
```

Diese Funktion wird bei den einzelnen Klassen nicht weiter erwähnt.

## 3.1 CMeister

Die Klasse `CMeister` besitzt außer dem Konstruktur zwei weitere Schnittstellenfunktionen:

**Konstruktor:**

```
CMeister::CMeister (string strName);
```

Der Konstruktor erwartet als Parameter den Namen des Spiels.

**Eintritt in die Menüschleife:**

```
virtual int CMeister::Start();
```

Die Startfunktion aktiviert die Menüschleife. Es werden keine weiteren Parameter benötigt, da diese alle im Objekt selbst gesetzt werden. Rückgabewert ist derzeit immer 0. Ein erweitertes Fehlermanagement könnte hier Werte an das aufrufende System zurückgeben.

**Überprüfung auf Spielende:**

```
virtual int CMeister::CheckEnde (CRaum * pRaum);
```

Diese Funktion überprüft, ob der übergebene Raum gleich dem definierten Zielraum ist. Dies ist derzeit die Gewinnbedingung.

## 3.2 CSpieler

`CSpieler` besitzt nur eine weitere Schnittstellenfunktion außer dem Konstruktor:

**Konstruktor:**

```
CSpieler::CSpieler (string strName, CMeister * pMeister);
```

Der Konstruktor initialisisert den Spieler und benötigt dafür den Namen des Spielers und einen Zeiger auf die `CMeister` - Klasse, zu der er zugeordnet ist. Dieser Zeiger wird zur Überprüfung der Gewinnbedingung genutzt.

**Eintritt in die Menüschleife**

```
virtual int CSpieler::Start (CRaum * pStart,
                             CList <CGegenstand *> * plistInventar);
```

Die Startfunktion aktiviert auch hier die Menüschleife. Der Spieler bekommt die Anfangsbedingungen Startraum und Startinventar beim Aufruf übergeben, mit deren Hilfe der Spieler sich durch das Labyrinth bewegt. Der Rückgabewert ist immer 0. Wie bei `CMeister` könnte auch hier ein Fehlermanagement eingeführt werden.

## 3.3 CRaum

`CRaum` besitzt als Datenobjekt naturgemäß mehrere Zugriffsfunktionen. Da diese für Gegenstände und Räume identisch sind, wird hier nur die Schnittstelle zur Raumverbindung erläutert.

**Konstruktor:**

```
CRaum::CRaum (string strName);
```

Der Konstruktor erstellt die notwendigen Listen und weist dem Raum den Namen `strName` zu.

**Raumverbindung hinzufügen:**

```
virtual int CRaum::AddRaum (CRaum * pRaum);
```

Diese Funktion hängt den übergebenen Raum an die Liste mit Raumverbindungen an und liefert 0 zurück. Mögliche Fehlerfälle existieren bislang nicht. Ist die Verbindung bereits vorhanden, wird die Anfrage ignoriert.

**Raumverbindung abfragen:**

```
virtual int CRaum::GetRaum (int iPos, CRaum * & pRaum);
```

`GetRaum(...)` liefert einen Zeiger auf den Raum, der durch die Verbindung `iPos` erreichbar ist, zurück. Ist der Index außerhalb des gültigen Bereiches, so ist der Rückgabewert der Funktion `1` und der zurückgelieferte `CRaum` - Zeiger ist ungültig.

**Raumverbindung entfernen:**

```
virtual int CRaum::RemoveRaum (int iPos, CRaum * & pRaum);
```

`GetRaum(...)` liefert einen Zeiger auf den Raum, der durch die Verbindung `iPos` erreichbar ist, zurück und entfernt diesen aus der Liste der vorhandenen Verbindungen. Ist der Index außerhalb des gültigen Bereiches, so ist der Rückgabewert der Funktion `1` und der zurückgelieferte `CRaum` - Zeiger ist ungültig.

## 3.4 CGegenstand

Die Klasse `CGegenstand` besitzt außer dem Konstruktor keine öffentlichen Zugriffsfunktionen.

**Konstruktor:**

```
CGegenstand::CGegenstand (string strName);
```

Der Konstruktor weißt dem Gegenstand den Namen `strName` zu. Weitere Initialisationen nicht notwendig, da keine weiteren Datenstrukturen bestehen.

# A Meister.h

```
////////////////////////////////////////////////////////////////////
//
//  Textadventure: Klasse zur Labyrinterstellung "CMeister"
//                 Schnittstelle
//
//  Datei:      Meister.h
//  Zweck:      Basisklasse fuer die Erstellung und Kontrolle des
//              Labyrinthes
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//  Version:    1.00
//  Datum:      24.11.1998
//
//  Implementierung in:
//              Meister.cpp
//
////////////////////////////////////////////////////////////////////


#if !defined(MEISTER_H)
#define MEISTER_H

// Includes
#include "CList.h"
#include "Gegenstand.h"
#include "Spieler.h"
#include "Raum.h"
#include <string>

// Forward - Deklaration der Klasse CSpieler, da CSpieler und CMeister
// sich gegenseitig verwenden.
class CSpieler;

class CMeister
{
private:
//////////////////////////////////////////////////////////////////////
// Verwaltungsdaten
//
  // Zeiger auf Spieler
    CSpieler * m_pSpieler;

  // Zeiger auf alle vorhandenen Raeume bzw. Gegenstaende. Wird fuer
  // Verwaltungszweke (Destruktor) benoetigt.
    CList<CRaum *> * m_plistRaum;
    CList<CGegenstand *> * m_plistGegenstand;

  // Zeiger auf aktuelles Spielerinventar.
    CList<CGegenstand *> * m_plistInventar;

  // Zeiger auf Start- und Endraum
    CRaum * m_praumBeginn;
    CRaum * m_praumEnde;

  // Name des Spieles
```

```
    string * m_pstrName;


////////////////////////////////////////////////////////////////////////
// Messagehandler fuer Start
//

  // Gegenstandsverwaltung
    virtual int msgGegenstandErstellen();
    virtual int msgGegenstandZuordnen();
    virtual int msgGegenstandLoeschen();
  // Raumverwaltung
    virtual int msgRaumErstellen();
    virtual int msgRaumZuordnen();
    virtual int msgRaumLoeschen();
  // Listenausgabe
    virtual int msgGegenstandListe();
    virtual int msgRaumListe();
  // Spielerverwaltung
    virtual int msgSpielerAnlegen();
    virtual int msgSpielerLoeschen();
  // Spielzustand
    virtual int msgStartEnde();
    virtual int msgSpielstart();

public:
////////////////////////////////////////////////////////////////////////
// Konstruktor / Destruktior
//
    CMeister(string strName);
    virtual ~CMeister();


////////////////////////////////////////////////////////////////////////
// Zugriffsfunktionen
//
    virtual string GetName ();

  // CheckEnde prueft, ob der Spieler sich im Zielraum befindet.
    virtual int CheckEnde (CRaum * pRaum);

  // Spiel starten
    virtual int Start ();

};

#endif // !defined(MEISTER_H)
```

# B Meister.cpp

```
//////////////////////////////////////////////////////////////////////
//
//  Textadventure: Klasse zur Labyrinterstellung "CMeister"
//                 Implementierung
//
//  Datei:      Meister.cpp
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//
//////////////////////////////////////////////////////////////////////

#include "stdio.h"
#include "iostream.h"
#include "Meister.h"
#include "stdlib.h"
#include <string>


//////////////////////////////////////////////////////////////////////
// Konstruktion/Destruktion
//////////////////////////////////////////////////////////////////////

CMeister::CMeister(string strName)
{
  // Der Konstruktor der Klasse CMeister erstellt alle Listen,
  // Initialisiert die Spielerabhaengigen Variablen mit NULL
  // und weist den Namen der Klasse zu.
    m_plistRaum = new CList<CRaum *>;
    m_plistGegenstand = new CList<CGegenstand *>;
    m_plistInventar = new CList<CGegenstand *>;

    m_pSpieler = NULL;
    m_praumBeginn = NULL;
    m_praumEnde = NULL;

    m_pstrName = new string(strName);

}

CMeister::~CMeister()
{
  // Der Destruktor loescht anhand der beiden Verwaltungslisten
  // alle vorhandenen Raeume und Gegenstaende und gibt somit
  // allen belegten Specherplatz wieder frei.
    CRaum * pRaum;
    CGegenstand * pGegenstand;

    while(!m_plistRaum->RemoveData(1, pRaum))
        delete pRaum;
    while(!m_plistGegenstand->RemoveData(1, pGegenstand))
        delete pGegenstand;

    delete m_plistRaum;
    delete m_plistGegenstand;
    delete m_plistInventar;
```

```
    if (m_pSpieler)
        delete m_pSpieler;
    delete m_pstrName;
}


int CMeister::GetName (string & strName)
{
    strName = *m_pstrName;
    return 0;
}


int CMeister::Start ()
{
  // Die Startfunktion stellt das Hauptmenue dar, aus dem verschiedene
  // Untermenuepunkte aufgerufen werden. Es findet keine Verarbeitung
  // innerhalb dieser Funktion statt.
    char ch;
    int loop = 1;

    while(loop)
    {
        cout << endl;
        cout << "**********************************"
             << "**************************\n";
        cout << "*Rollenspiel - Testengine          "
        cout << "**********************************"
             << "**************************\n\n";
        cout << "[ 1 ] Raum erstellen\n";
        cout << "[ 2 ] Raum loeschen\n";
        cout << "[ 3 ] Raum zuordnen\n";
        cout << "[ 4 ] Gegenstand erstellen\n";
        cout << "[ 5 ] Gegenstand loeschen\n";
        cout << "[ 6 ] Gegenstand zuordnen\n";
        cout << "[ 7 ] Start- und Endraum waehlen\n";
        cout << "[ 8 ] Spieler anlegen\n";
        cout << "[ 9 ] Spieler loeschen\n";
        cout << "[ 0 ] Testspiel starten\n";
        cout << "[ R ] Raumliste anzeigen\n";
        cout << "[ G ] Gegenstandsliste anzeigen\n";
        cout << "[ X ] Beenden\n\n";
        cout << "Choose your destination: ";


        cin >> ch;

        switch(toupper(ch))
        {
        case '1':
            msgRaumErstellen();
            break;
        case '2':
            msgRaumLoeschen();
            break;
        case '3':
            msgRaumZuordnen();
            break;
        case '4':
            msgGegenstandErstellen();
```

```
            break;
        case '5':
            msgGegenstandLoeschen();
            break;
        case '6':
            msgGegenstandZuordnen();
            break;
        case '7':
            msgStartEnde();
            break;
        case '8':
            msgSpielerAnlegen();
            break;
        case '9':
            msgSpielerLoeschen();
            break;
        case '0':
            msgSpielstart();
            break;
        case 'R':
            msgRaumListe();
            break;
        case 'G':
            msgGegenstandListe();
            break;
        case 'X':
            loop = 0;
            break;
        default:
            break;
        }
    }
}

int CMeister::msgRaumErstellen()
{
  // Diese Funktion erstellt einen neuen Raum, dessen Name
  // vom Benutzer festgelegt wird. Der neue Raum besitzt
  // zu diesem Zeitpunkt noch keine weiteren Verbindungen
  // oder Gegenstaende.
    CRaum * pNew;
    string str;

    cout << "\n\n\n";
    cout << "Raum anlegen:\n";
    cout << "=============\n";
    cout << "Bitte geben sie den Namen des neuen Raumes an: ";
    cin >> str;

    pNew = new CRaum (str);
    m_plistRaum->AddData(pNew);

    return 0;
}

int CMeister::msgRaumLoeschen()
{
  // Anhand der Raumliste wird der zu loeschende Raum ausgewaehlt.
  // Vor dem endgueltigen Loeschvorgang wird geprueft, ob noch
  // Beziehungen zwischen anderen Objekten und dem zu loeschendem
```

```
  // Raum bestehen.
    CRaum * pDel;
    CRaum * pTemp;
    CRaum * pTemp2;
    int i,j;

    if (m_plistRaum->IsEmpty())
    {
        cout << "\n\n\nDie Liste enthält keine Elemente!\n";
        return 1;
    }

    cout << "\n\n\n";
    cout << "Raum loeschen:\n";
    cout << "==============\n";
    msgRaumListe();
    cout << "Nummer des zu loeschenden Raumes eingeben: ";
    cin >> i;
    cout << endl;

    if (!m_plistRaum->RemoveData(i, pDel))
        delete pDel;
    else
        return 2;

    i = 0;
    while(!m_plistRaum->GetData(++i, pTemp))
    {
        j = 0;
        while(!pTemp->GetRaum(++j, pTemp2))
        {
            if (pTemp2 == pDel)
            {
                pTemp->RemoveRaum(j, pTemp2);
                j--;
            }
        }
    }
    return 0;
}

int CMeister::msgRaumZuordnen()
{
  // Mit dieser Funktion wird der Benutzer dazu aufgefordert, die
  // Verbindung zwischen zwei Raeumen herzustellen.
    CRaum * pSource;
    CRaum * pDest;
    int i;

    if (m_plistRaum->IsEmpty())
    {
        cout << "Es waere ratsam, zuerst einen Raum anzulegen...\n";
        return 2;
    }

    cout << "\n\n\n";
    cout << "Raum zuordnen:\n";
    cout << "==============\n";
    msgRaumListe();
    cout << "Bitte waehlen Sie die Nummer des Ursprungsraums"
```

```
         << " der Verbindung: ";
    cin >> i;
    cout << endl;
    if (!m_plistRaum->GetData(i, pSource))
        cout << "Herzlichen Glueckwunsch, dieser Raum "
             << "existiert tatsaechlich!\n";
    else
    {
        cout << "Herzlichen Glueckwunsch, dieser Raum "
             << "existiert LEIDER NICHT!\n";
        return 1;
    }

    msgRaumListe();
    cout << "Bitte waehlen Sie die Nummer des Zielraums der Verbindung: ";
    cin >> i;
    cout << endl;
    if (!m_plistRaum->GetData(i, pDest))
        cout << "Herzlichen Glueckwunsch, auch dieser Raum "
             << "existiert tatsaechlich!\n";
    else
    {
        cout << "Herzlichen Glueckwunsch, dieser Raum "
             << "existiert LEIDER NICHT!\n";
        return 2;
    }

    pSource->AddRaum(pDest);
    return 0;
}

int CMeister::msgGegenstandErstellen()
{
  // Diese Funktion legt einen neuen Gegenstand an. Er ist vorlaeufig
  // noch keinem Objekt zugeordnet.
    CGegenstand * pNew;
    string str;

    cout << "\n\n\n";
    cout << "Gegenstand anlegen:\n";
    cout << "===================\n";
    cout << "Bitte geben sie den Namen des neuen Gegenstandes an: ";
    cin >> str;

    pNew = new CGegenstand (str);
    m_plistGegenstand->AddData(pNew);

    return 0;

}

int CMeister::msgGegenstandLoeschen()
{
  // Ein Gegenstand wird geloescht. Vorhandene Beziehungen werden gesucht
  // und, falls vorhanden, entfernt.
    CGegenstand * pDel;
    CRaum * pTemp;
    CGegenstand * pTemp2;

    int i,j;
```

```
    if (m_plistGegenstand->IsEmpty())
    {
        cout << "\n\n\nDie Liste enthält keine Elemente!\n";
        return 1;
    }

    cout << "\n\n\n";
    cout << "Gegenstand loeschen:\n";
    cout << "====================\n";
    msgGegenstandListe();
    cout << "Nummer des zu loeschenden Gegenstandes eingeben: ";
    cin >> i;
    cout << endl;

    if (!m_plistGegenstand->RemoveData(i, pDel))
        delete pDel;

    else
        return 2;

    i = 1;
    while(!m_plistRaum->GetData(i, pTemp))
    {
        j = 1;
        while(!pTemp->GetGegenstand(j, pTemp2))
        {
            if (pTemp2 == pDel)
            {
                pTemp->RemoveGegenstand(j, pTemp2);
                j--;
            }
            j++;
        }
        i++;
    }

    i = 1;
    while (!m_plistInventar->GetData(i, pTemp2))
    {
        if (pTemp2 == pDel)
        {
            m_plistInventar->RemoveData(i, pTemp2);
            i--;
        }
        i++;
    }

    return 0;
}

int CMeister::msgGegenstandZuordnen()
{
  // Diese Funktion ordnet einen Gegenstand entweder einem Raum oder einem
  // Spieler zu. Existiert die Zuordnung bereits, so wird der Versuch
  // ignoriert.
    CGegenstand * pSource;
    CGegenstand * pGegenstand;
    CRaum * pDest;
    int i;
```

```
    char ch;

    cout << "\n\n\n";
    cout << "Gegenstand zuordnen:\n";
    cout << "====================\n";
    msgGegenstandListe();
    cout << "Bitte waehlen Sie die Nummer des Gegenstandes: ";
    cin >> i;
    cout << endl;
    if (!m_plistGegenstand->GetData(i, pSource))
        cout << "Herzlichen Glueckwunsch, dieser "
             << "Gegenstand existiert tatsaechlich!\n";
    else
    {
        cout << "Herzlichen Glueckwunsch, dieser "
             << "Gegenstand hat sich wohl in Rauch AUFGELOEST!\n";
        return 1;
    }

    cout << "Wenn Sie den Gegenstand einem Spieler zuordnen "
         << "wollen, dann druecken sie JETZT ein [J]: ";
    cin >> ch;

    if (toupper(ch) == 'J')
    {
        i = 0;
        while (!m_plistInventar->GetData(++i, pGegenstand))
            if (pGegenstand == pSource)
                return 0;

        m_plistInventar->AddData(pSource);
        return 0;
    }

    if (m_plistRaum->IsEmpty())
    {
        cout << "Es waere ratsam, zuerst einen Raum anzulegen...\n";
        return 2;
    }

    msgRaumListe();
    cout << "Bitte waehlen Sie die Nummer des Raumes: ";
    cin >> i;
    cout << endl;
    if (!m_plistRaum->GetData(i, pDest))
        cout << "Herzlichen Glueckwunsch, auch dieser Raum "
             << "existiert tatsaechlich!\n";
    else
    {
        cout << "Herzlichen Glueckwunsch, dieser Raum "
             << "existiert LEIDER NICHT!\n";
        return 3;
    }

    i = 0;
    while (!pDest->GetGegenstand(++i, pGegenstand))
        if (pGegenstand == pSource)
            return 0;
```

```
    pDest->AddGegenstand(pSource);
    return 0;

}

int CMeister::msgStartEnde()
{
  // Mit dieser Funktion wird der Start- und Endraum festgelegt.
    cout << "\n\n\n";
    cout << "Start- und Endraum festlegen:";
    cout << "============================";

    if(m_plistRaum->IsEmpty())
    {
        cout << "Es waere ratsam zuerst ein paar "
             << "Raeume anzulegen...\n";
        return 1;
    }

    CRaum * pStart;
    CRaum * pEnde;
    int i;

    msgRaumListe();
    cout << "Bitte geben Sie die Nummer des Startraumes ein: ";
    cin >> i;
    cout << endl;

    if (!m_plistRaum->GetData(i, pStart))
        cout << "Herzlichen Glueckwunsch, sie haben "
             << "einen existierenden Raum getroffen!\n";
    else
    {
        cout << "Diese Raumnummer existiert nicht!\n";
        return 2;
    }

    msgRaumListe();
    cout << "Bitte geben Sie die Nummer des Zielraumes ein: ";
    cin >> i;
    cout << endl;

    if (!m_plistRaum->GetData (i, pEnde))
        cout << "Herzlichen Glueckwunsch, auch dieser "
             << "Raum existiert! Ob er Zugaenge
            hat,\nsteht auf einem anderen Blatt\n";
    else
    {
        cout << "Diese Raumnummer existiert nicht!\n";
        return 3;
    }

    m_praumBeginn = pStart;
    m_praumEnde = pEnde;

    cout << "Genial. Sie haben die Zuweisung eines Start- "
         << "und Endraumes erfolgreich\nvollzogen!\n";
    return 0;

}
```

```
int CMeister::msgSpielerAnlegen()
{
  // Diese Funktion erstellt den Spieler. Ist bereits ein Spieler angelegt,
  // so bricht die Funktion ab.
    string str;

    cout << "\n\n\n";
    cout << "Spieler anlegen:\n";
    cout << "================\n";

    if (m_pSpieler)
    {
        cout << "Es ist bereits ein Spieler angelegt. "
             << "Toeten Sie diesen zuerst!\n";
        return 1;
    }

    cout << "Bitte geben Sie den Namen des Spielers ein. "
         << "Aehnlichkeiten mit Lebenden\n"
         << "Personen sind nicht ausgeschlossen, aber rein zufaellig: ";
    cin >> str;

    m_pSpieler = new CSpieler(str, this);

    return 0;
}

int CMeister::msgSpielerLoeschen()
{
  // Diese Funktion loescht, derzeit ohne Sicherheitsabfrage, den
  // aktuellen Spieler.
    cout << "\n\n\n";
    cout << "Spieler toeten:\n";
    cout << "===============\n";

    if (!m_pSpieler)
    {
        cout << "Beschaeftigst Du Dich immer mit Schattenboxen? "
             << "Hier existiert niemand, der kaputtbar waere...\n";
        return 1;
    }
    cout << "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
         << "RRRRRRRRRRRRRRRRRRRRRRGHHHHHHHH!\n";
    cout << "<plumps> Ein regloser Koerper schlaegt "
         << "auf dem Boden auf und wird vom\n"
         << "Aufraeumdienst nach NIL verschoben.\n";

    cout << "\nWaren Sie sich auch wirklich sicher !?!\n";
    delete m_pSpieler;
    m_pSpieler = NULL;
    return 0;
}

int CMeister::msgSpielstart()
{
  // Diese Funktion startet das Spiel, wenn alle noetigen
  // Datenstrukturen angelegt sind.
    if (!m_pSpieler)
    {
```

```
        cout << "Es existiert kein Spieler!\n";
        return 1;
    }
    if (!m_praumBeginn || !m_praumEnde)
    {
        cout << "Start- oder Endraum nicht definert!\n";
        return 2;
    }

    CList<CGegenstand *> * plistTemp = new CList<CGegenstand *>;
    CGegenstand * pGegenstand;

    while (!m_plistInventar->RemoveData(1, pGegenstand))
        plistTemp->AddData(pGegenstand);

    m_pSpieler->Start(m_praumBeginn, plistTemp);

    while (!plistTemp->RemoveData(1, pGegenstand))
        m_plistInventar->AddData(pGegenstand);

    delete plistTemp;
    return 0;
}

int CMeister::msgRaumListe()
{
  // Hilfsfunktion, die alle vorhandenen Raeume ausgiebt.
  // Sie wird sowohl als eigenstaendige Funktion als auch
  // als Hilfsfunktion von einigen anderen Funktionen aus
  // aufgerufen.
    int iCount = 0;
    int iLine = 2;
    CRaum * pTemp;
    char ctemp;

    cout << "\n\n\n";

    cout << "Liste der Raeume:\n";
    cout << "=================\n";

    while (!m_plistRaum->GetData(++iCount, pTemp))
    {
        cout << iCount << ": " << pTemp->GetName() << endl;
        iLine++;
        if (iLine >= 23)
        {
            cout << "Eine Taste fuer weiter...\n";
            iLine = 0;
            cin >> ctemp;
        }
    }
    cout << "Eine Taste fuer weiter...\n";
    cin >> ctemp;
    return 0;
}

int CMeister::msgGegenstandListe()
{
  // Hilfsfunktion, die alle vorhandenen Gegenstaende ausgiebt.
  // Sie wird sowohl als eigenstaendige Funktion als auch
```

```
  // als Hilfsfunktion von einigen anderen Funktionen aus
  // aufgerufen.
    int iCount = 0;
    int iLine = 2;
    char ctemp;
    CGegenstand * pTemp;

    cout << "\n\n\n";

    cout << "Liste der Gegenstaende:\n";
    cout << "======================\n";

    while (!m_plistGegenstand->GetData(++iCount, pTemp))
    {
        cout << iCount << ": " << pTemp->GetName() << endl;
        iLine++;
        if (iLine >= 23)
        {
            cout << "Eine Taste fuer weiter...\n";
            iLine = 0;
            cin >> ctemp;
        }
    }
    cout << "Eine Taste fuer weiter...\n";
    cin >> ctemp;
    return 0;
}

int CMeister::CheckEnde (CRaum * pRaum)
{
    return (pRaum == m_praumEnde);
}
```

# C Spieler.h

```
////////////////////////////////////////////////////////////////////
//
//  Textadventure: Spielerklasse "CSpieler"
//
//  Datei:      Spieler.h
//  Zweck:      Basisklasse fuer das eigentliche Text-Adventure
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//  Version:    1.00
//  Datum:      24.11.1998
//
//  Implementierung in:
//              Spieler.cpp
//
////////////////////////////////////////////////////////////////////


#if !defined(SPIELER_H)
#define SPIELER_H

// Includes
#include "Meister.h"
#include "CList.h"
#include "Gegenstand.h"
#include "Raum.h"
#include <string>

// Forward - Deklaration der Klasse CMeister, da CSpieler und CMeister
// sich gegenseitig verwenden.
class CMeister;

class CSpieler
{

private:
////////////////////////////////////////////////////////////////////
// Verwaltungsdaten
//
  // Aktueller Raum
    CRaum * m_praumAkt;
  // Aktuelles Inventar
    CList<CGegenstand *> * m_plistGegenstand;
  // Name des Spielers
    string * m_pstrName;
  // Zeiger auf Meister, der den Spieler erstellt hat.
    CMeister * m_pMeister;

////////////////////////////////////////////////////////////////////
// Private Hilfsfunktionen
//
    int TakeGegenstand (int iPos);
    int DropGegenstand (int iPos);
    int LeaveRaum (int iPos);

////////////////////////////////////////////////////////////////////
```

```
// Messagehandler fuer Start
//
  // Listenausgaben
    virtual int msgShowWege();
    virtual int msgShowGegenstaende();
    virtual int msgShowInventar();
  // Kommandos
    virtual int msgMove();
    virtual int msgLook();
    virtual int msgTakeGegenstand();
    virtual int msgDropGegenstand();



////////////////////////////////////////////////////////////////////////
// Konstruktor / Destruktior
//
public:
    CSpieler (string strName, CMeister * pMeister);
    virtual ~CSpieler();

////////////////////////////////////////////////////////////////////////
// Zugriffsfunktionen
//

    virtual string GetName ();

  // Spiel starten
    virtual int Start (CRaum * pStart,
                       CList<CGegenstand *> * plistInventar);



};

#endif // !defined(SPIELER_H)
```

# D Spieler.cpp

```
//////////////////////////////////////////////////////////////////////
//
//  Textadventure: Spielerklasse "CSpieler" - Implementierung
//
//  Datei:      Gegenstand.cpp
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//
//////////////////////////////////////////////////////////////////////

#include "stdio.h"
#include "iostream.h"
#include "Spieler.h"

CSpieler::CSpieler (string strName, CMeister * pMeister)
{
  // Die Spielerinternen Variablen werden initialisiert.
    m_pstrName = new string;
    *m_pstrName = strName;

    m_praumAkt = NULL;

    m_plistGegenstand = new CList<CGegenstand *>;

    m_pMeister = pMeister;
}

CSpieler::~CSpieler ()
{
  // Alle angelegten Variablen werden wieder freigegeben.
    delete m_pstrName;
    delete m_plistGegenstand;
}

string CSpieler::GetName ()
{
  // Liefert den Namen des aktuellen Spielers zurück.
    return (*m_pstrName);
}

int CSpieler::TakeGegenstand (int iPos)
{
  // Private Hilfsfunktion, die den Gegenstand mit der Nummer
  // iPos aus dem aktuellen Raum aufnimmt.
  // Fehlercode "1": Aktueller Raum ist nicht definiert
  // Fehlercode "2": Gegenstandsindex ausserhalb des gültigen Bereichs.
    CGegenstand * pTemp;
    int iError;

    if (!m_praumAkt)
        // Aktueller Raum nicht definiert
        return 1;

    if (m_praumAkt->RemoveGegenstand(iPos, pTemp))
        // Gegenstandsindex ausserhalb des gueltigen Bereichs
```

```
        return 2;

    m_plistGegenstand->AddData(pTemp);

    return 0;
}

int CSpieler::DropGegenstand (int iPos)
{
  // Private Hilfsfunktion, die den Gegenstand mit der Nummer
  // iPos in den aktuellen Raum ablegt.
  // Fehlercode "1": Aktueller Raum ist nicht definiert
  // Fehlercode "2": Gegenstandsindex ausserhalb des gültigen Bereichs.
    CGegenstand * pTemp;

    if (!m_praumAkt)
        // Aktueller Raum nicht definiert
        return 1;

    if (m_plistGegenstand->RemoveData (iPos, pTemp))
        // Gegenstandsindex ausserhalb des gueltigen Bereichs
        return 2;

    m_praumAkt->AddGegenstand(pTemp);

    return 0;
}

int CSpieler::LeaveRaum (int iPos)
{
  // Private Hilfsfunktion, die den aktuellen Raum über den Weg iPos
  // verläßt.
  // Fehlercode "1": Spiel ist gewonnen. Der aktuelle raum wurde
  //                 aber gewechselt!
  // Fehlercode "2": Der aktuelle Raum ist nicht definiert.
  // Fehlercode "3": Der Index ist außerhalb des gültigen Bereichs.
    CRaum * pTemp;
    int iError;

    if (!m_praumAkt)
        // Aktueller Raum nicht definiert
        return 2;

    if (m_praumAkt->GetRaum (iPos, pTemp))
        // Raumindex ausserhalb des gueltigen Bereichs
        return 3;

    m_praumAkt = pTemp;

    return (m_pMeister->CheckEnde(m_praumAkt) ? 1 : 0);
}

virtual int  CSpieler::Start (CRaum * pStart,
                              CList<CGegenstand *> * plistInventar)
{
  // Die Startfunktion kopiert das übergebene Inventar in das des
  // Spielers, startet die Menüschleife und kopiert das Spielerinterne
  // Inventar vor dem beenden der Funktion wieder zurück in die übergebene
  // Liste.
    m_praumAkt = pStart;
```

```
    CGegenstand * pGegenstand;
    char ch;
    int loop = 1;

    while (!plistInventar->RemoveData(1, pGegenstand))
        m_plistGegenstand->AddData(pGegenstand);

    cout << "\n\n\n\n";
    cout << "==========\n";
    cout << "SPIELSTART\n";
    cout << "==========\n";
    cout << "\n\n\n\n";

    while(loop)
    {
        cout << "Aktueller Raum: " << m_praumAkt->GetName() << "\n";
        cout << "[B]ewegen, [W]ege, [G]egenstaende, [I]nventar, "
             << "[U]msehen, [N]ehmen, [A]blegen, [E]nde\n";
        cin >> ch;
        switch(toupper(ch))
        {
        case 'B':
            msgMove();
            if(m_pMeister->CheckEnde(m_praumAkt))
                loop = 0;
            break;
        case 'W':
            msgShowWege();
            break;
        case 'G':
            msgShowGegenstaende();
            break;
        case 'I':
            msgShowInventar();
            break;
        case 'U':
            msgLook();
            break;
        case 'N':
            msgTakeGegenstand();
            break;
        case 'A':
            msgDropGegenstand();
            break;
        case 'E':
            loop = 0;
            break;
        }
    }

    while (!m_plistGegenstand->RemoveData(1, pGegenstand))
        plistInventar->AddData(pGegenstand);

    return 0;
}

void CSpieler::msgShowWege()
{
  // Diese Listenfunktion zeigt alle vorhandenen Wege in andere Räume an.
    int i = 0;
```

```
    int c = 1;
    char ch;
    CRaum * pRaum;

    cout << "\n";
    cout << "Verbluefft schaust Du Dich um und siehst folgende Wege:\n";

    while (!m_praumAkt->GetRaum(++i, pRaum))
    {
        cout << "   " << i << ": " << pRaum->GetName() << "\n";
        c++;
        if (c > 22)
        {
            cout << "Press almost any key to continue...\n";
            cin >> ch;
        }
    }
    cout << "\n";
}

void CSpieler::msgShowGegenstaende()
{
  // Diese Listenfunktion zeigt alle vorhandenen Gegenstände im aktuellen
  // Raum an.
    int i = 0;
    int c = 1;
    char ch;
    CGegenstand * pGegenstand;

    cout << "\n";
    cout << "Verbluefft schaust Du Dich um und siehst "
         << "folgende Gegenstaende:\n";

    while (!m_praumAkt->GetGegenstand(++i, pGegenstand))
    {
        cout << "   " << i << ": " << pGegenstand->GetName() << "\n";
        c++;
        if (c > 22)
        {
            cout << "Press almost any key to continue...\n";
            cin >> ch;
        }
    }
    cout << "\n";
}

void CSpieler::msgShowInventar()
{
  // Diese Listenfunktion zeigt alle vorhandenen Gegenstände im Spieler-
  // Inventar an.
    int i = 0;
    int c = 1;
    char ch;
    CGegenstand * pGegenstand;

    cout << "\n";
    cout << "Du findest in Deinen Taschen folgende Gegenstaende:\n";

    while (!m_plistGegenstand->GetData(++i, pGegenstand))
    {
```

```
        cout << "   " << i << ": " << pGegenstand->GetName() << "\n";
        c++;
        if (c > 22)
        {
            cout << "Press almost any key to continue...\n";
            cin >> ch;
        }
    }
    cout << "\n";
}


void CSpieler::msgMove()
{
  // Diese Funktion fordert den Spieler auf die Eingabe einer Wegnummer
  // auf und übergiebt dies anschließend der Funktion LeaveRaum um den
  // Spieler in den neuen Raum zu bewegen.
    int i;

    msgShowWege();

    cout << "Taxi - Scharia.... Wo Du wollen ?";
    cin >> i;
    cout << "\n";

    switch(LeaveRaum (i))
    {
    case 3:
    case 2:
        cout << "Zentrale, hier Uetzwurscht. Wo sein dieses Raum.";
        break;
    case 1:
      // Spiel gewonnen!
        cout << "Elvis leben! Wir fahre Memphis!";
        break;
    case 0:
        cout << "Nix Memphis!";
        break;
    }
}

void CSpieler::msgLook()
{
  // Diese Funktion zeigt das komplette Umfeld des Spielers an.
    char ch;
    msgShowWege();
    cout << "\nPress almost any key to continue...\n";
    cin >> ch;
    msgShowGegenstaende();
    cout << "\nPress almost any key to continue...\n";
    cin >> ch;
    msgShowInventar();
    cout << "\nPress almost any key to continue...\n";
    cin >> ch;
}

void CSpieler::msgTakeGegenstand()
{
  // Der Benutzer wird zur Angabe eines Gegenstandes aufgefordert,
  // der danach mittels TakeGegenstandes ins Spielerinventar aufgenommen
```

```
  // wird.
    int i;

    msgShowGegenstaende();

    cout << "Welchen Gegenstand soll ich aufheben ?";
    cin >> i;
    cout << "\n";

    switch(TakeGegenstand (i))
    {
    case 2:
    case 1:
        cout << "Diesen Gegenstand gibt es nicht!";
        break;
    case 0:
        cout << "OKai";
        break;
    }
}

void CSpieler::msgDropGegenstand()
{
  // Der Benutzer wird zur Angabe eines Gegenstandes aufgefordert,
  // der danach mittels TakeGegenstandes aus dem Spielerinventar
  // entfernt wird.
    int i;

    msgShowInventar();

    cout << "Welchen Gegenstand soll ich ablegen ?";
    cin >> i;
    cout << "\n";

    switch(DropGegenstand (i))
    {
    case 2:
    case 1:
        cout << "Diesen Gegenstand gibt es nicht!";
        break;
    case 0:
        cout << "OKai";
        break;
    }
}
```

# E Raum.h

```
//////////////////////////////////////////////////////////////////////
//
//  Textadventure: Raumsklasse "CRaum"
//
//  Datei:      Raum.h
//  Zweck:      Basisklasse fuer Raumverwaltung im Text-Adventure
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//  Version:    1.00
//  Datum:      24.11.1998
//
//  Implementierung in:
//              Raum.cpp
//
//////////////////////////////////////////////////////////////////////

#if !defined(RAUM_H)
#define RAUM_H

// Includes
#include "CList.h"
#include "Gegenstand.h"
#include <string>

class CRaum
{
private:
//////////////////////////////////////////////////////////////////////
// Verwaltungsdaten
//
  // Zeiger auf vorhandene Raumverbindungen und Gegenstände
    CList<CRaum *> * m_plistRaum;
    CList<CGegenstand *> * m_plistGegenstand;
  // Name des Raumes
    string * m_pstrName;

public:
//////////////////////////////////////////////////////////////////////
// Konstruktor / Destruktior
//
    CRaum(string strName);
    virtual ~CRaum();

//////////////////////////////////////////////////////////////////////
// Zugriffsfunktionen
//
    virtual string GetName ();

// Zugriffsfunktionen für angrenzende Räume
    int AddRaum (CRaum * pRaum);
    int GetRaum (int iPos, CRaum * & pRaum);
    int RemoveRaum (int iPos, CRaum * & pRaum);

// Zugriffsfunktionen für Gegenstände
    int AddGegenstand (CGegenstand * pGegenstand);
```

```
    int GetGegenstand (int iPos, CGegenstand * & pGegenstand);
    int RemoveGegenstand (int iPos, CGegenstand * & pGegenstand);

};

#endif // !defined(RAUM_H)
```

# F Raum.cpp

```
////////////////////////////////////////////////////////////////////
//
//  Textadventure: Raumklasse "CRaum" - Implementierung
//
//  Datei:      Raum.cpp
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//
////////////////////////////////////////////////////////////////////


#include "CList.h"
#include "Raum.h"
#include "Gegenstand.h"

CRaum::CRaum(string strName)
{
  // Der Konstruktor initialisiert die Listen und Membervariabeln
  // der Klasse.
    m_pstrName = new string (strName);
    m_plistGegenstand = new CList<CGegenstand *>;
    m_plistRaum = new CList<CRaum *>;
}

CRaum::~CRaum()
{
  // Der Destruktor löscht die angelegten Objekte wieder.
    delete m_pstrName;
    delete m_plistGegenstand;
    delete m_plistRaum;
}


tring CRaum::GetName ()
{
  // GetName liefert den Namen des Objektes zurück
    return (*m_pstrName);
}

int CRaum::AddRaum (CRaum * pRaum)
{
  // Ein Raum wird in die Liste eingefügt.
    m_plistRaum->AddData(pRaum);
    return 0;
}

int CRaum::GetRaum (int iPos, CRaum * & pRaum)
{
  // Ein Raum wird aus der Liste ausgelesen.
  // Negation aufgrund von Vereinbarungen der Fehlerwerte
  // da die Liste den Wert 0 als Fehler anninmt.
    return m_plistRaum->GetData(iPos, pRaum);
}

int CRaum::RemoveRaum (int iPos, CRaum * & pRaum)
```

```
{
  // Ein Raum wird aus der Liste ausgelesen und entfernt.
  // Negation aufgrund von Vereinbarungen der Fehlerwerte
  // da die Liste den Wert 0 als Fehler anninmt.
    return m_plistRaum->RemoveData(iPos, pRaum);
}

int CRaum::AddGegenstand (CGegenstand * pGegenstand)
{
  // Ein Gegenstand wird in die Liste eingefügt.
    m_plistGegenstand->AddData(pGegenstand);
    return 0;
}

int CRaum::GetGegenstand (int iPos, CGegenstand * & pGegenstand)
{
  // Ein Gegenstand wird aus der Liste ausgelesen.
  // Negation aufgrund von Vereinbarungen der Fehlerwerte
  // da die Liste den Wert 0 als Fehler anninmt.
    return m_plistGegenstand->GetData(iPos, pGegenstand);
}

int CRaum::RemoveGegenstand (int iPos, CGegenstand * & pGegenstand)
{
  // Ein Gegenstand wird aus der Liste ausgelesen und entfernt.
  // Negation aufgrund von Vereinbarungen der Fehlerwerte
  // da die Liste den Wert 0 als Fehler anninmt.
    return m_plistGegenstand->RemoveData(iPos, pGegenstand);
}
```

# G Gegenstand.h

```
//////////////////////////////////////////////////////////////////////
//
//  Textadventure: Gegenstandsklasse "CGegenstand"
//
//  Datei:      Gegenstand.h
//  Zweck:      Basisklasse fuer Gegenstandsverwaltung im Text-
//              Adventure
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//  Version:    1.00
//  Datum:      24.11.1998
//
//  Implementierung in:
//              Gegenstand.cpp
//
//////////////////////////////////////////////////////////////////////

#if !defined(GEGENSTAND_H)
#define GEGENSTAND_H

#include <string>

class CGegenstand
{

private:
    string * m_pstrName;


public:
//////////////////////////////////////////////////////////////////////
// Konstruktor / Destruktior
//

    CGegenstand(string strName);
    virtual ~CGegenstand();

//////////////////////////////////////////////////////////////////////
// Zugriffsfunktionen
//
    virtual string GetName();

};

#endif // !defined(GEGENSTAND_H)
```

# H Gegenstand.cpp

```
////////////////////////////////////////////////////////////////////
//
//  Textadventure: Gegenstandsklasse "CGegenstand" - Implementierung
//
//  Datei:      Gegenstand.cpp
//  Autoren:    Torben Nehmer <Torben.Nehmer@gmx.net>
//              Wolfgang Nitz <nitz@alpha.fh-furtwangen.de>
//              Thomas Graf <ThomasGraf@swol.de>
//              Peter Allgaier <P.Allgaier@vega-g.de>
//              Nina Siraky <siraky@foo.fh-furtwangen.de>
//
////////////////////////////////////////////////////////////////////

#include "Gegenstand.h"

CGegenstand::CGegenstand(string strName)
{
    m_pstrName = new string;
    *m_pstrName = strName;
}

CGegenstand::~CGegenstand()
{
    delete m_pstrName;
}

string CGegenstand::GetName()
{
    return *m_pstrName;
}
```

# I Beispielprogramm main.cpp

```
#include <stdio.h>
#include <iostream.h>
#include "Meister.h"

int main()
{

    CMeister * pMeister = new CMeister("TAXI SCHARIA");

    pMeister->Start();

    delete pMeister;

    return 0;
}
```