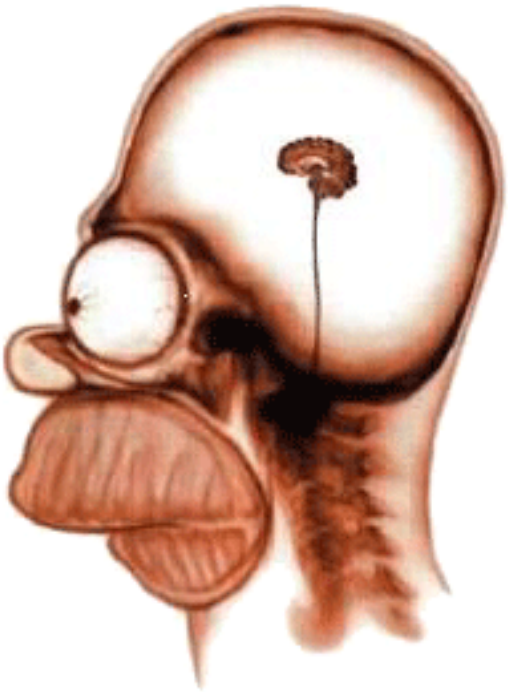

Kleinprojekt Mastermind

Gabi Rohner	Rainer Meier	Stephan Krattiger
Im Mattler 8	Käserei	Weidentalweg 14
8911 Rifferswil	6288 Schongau	4436 Oberdorf

28. Januar 2004

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Hochschule für Technik und Architektur fordert im Studiengang Informatik im Fach Programmieren die Durchführung eines Kleinprojektes. In diesem Kleinprojekt handelt es sich um eine anspruchsvollere Problemstellung als es die Aufgaben während den ersten beiden Semestern bisher beinhaltet haben. Anhand dieses Projektes erlernen die Studentinnen und Studenten die Planung und Realisierung objektorientierter Software im kleineren Rahmen, den Umgang mit den Dokumentationswerkzeugen UML und JavaDoc, der Bau eines eigenen GUIs, die Entwicklung eigener Algorithmen und allenfalls das File-Handling, die Objekt-Serialisierung und das MVC-Konzept.

Kapitel 2

Mastermind - das Spiel

2.1 Allgemeines

Mastermind ist ein Spiel für zwei Spieler. Ein Spieler gibt ein Codewort (Kombination von Farben) vor und der andere versucht dieses Codewort in einer vorgegebenen Anzahl Versuchen zu erraten. Der vorgebende Spieler bewertet dabei jeden Versuch des ratenden Spielers nach seiner Korrektheit mit schwarzen und weissen Pins.

2.2 Bewertungsregeln

Schwarzer Pin: richtige Farbe an der richtigen Stelle erraten
Weisser Punkt: richtige Farbe aber an der falschen Stelle erraten

Dabei hat die Reihenfolge der gegebenen Punkte nichts mit der Reihenfolge im geratenen oder vorgegebenen Codewort zu tun. Jede Farbe in der Vorgabe und im Versuch darf nur einmal zur Bewertung herangezogen werden. Ist das Codewort geknackt, so wird dies durch vier schwarze Pins gekennzeichnet.

2.3 Spielablauf

Der Computer gibt ein zufälliges, dem gewählten Schwierigkeitsgrad entsprechendes Codewort vor. Anschliessend kann der Spieler seine Versuche starten und vom Computer bewerten lassen.

Kapitel 3

Anforderungen

3.1 Minimale Anforderungen

Die minimalen Anforderungen beschreiben die Projektvorgaben des Dozenten. Sie sind zwingen zu implementieren. Minimale Anforderungen sind alle Grundfunktionen / Vorgaben, die nötig sind, dass man Mastermind überhaupt spielen kann.

3.1.1 Vorgaben

- Anzahl Farben: 6
- Anzahl Versuche: 8
- Länge des Codeworts: 4
- Mehrfachfarben nicht erlaubt
- Korrekturreihenfolge entspricht Pinreihenfolge

3.1.2 Funkionen

- Computer gibt zufälliges Codewort vor
- Neues Spiel
- Spiel beenden
- Pin setzen
- Gestecktes Codewort auswerten
- Codewort aufdecken

3.2 Erweiterte Anforderungen

Die erweiterten Vorgaben wurden teilweise vom Dozenten angedacht und als "Nice to have" eingestuft. Ebenso beinhalten sie die Ideen und Anregungen der Projektmitglieder.

3.2.1 Vorgaben

- Mehrfachfarben erlaubt
- Korrekturreihenfolge entspricht nicht Pinreihenfolge
- Zeitlimite für Rateversuche
- Multiplayer

3.2.2 Funktionen

- Spielmode änderbar (Einfach-/Mehrfachfarben)
- Anzeige und Berechnung eines Scores (Berechnung aus Spielmode und Anzahl Versuche)
- Speichern des High-Scores (z.B. Top-Ten)
- Spieleranzahl änderbar (Single-/Multiplayer)
- Sprache wählbar
- Hangman-Animation passend zu den noch verbleibenden Anzahl Versuchen

Kapitel 4

Analyse / Design

4.1 Use Cases

Ein Anwendungsfall (Use Case) ist die typische Beschreibung einer grundlegenden Interaktion zwischen dem Anwender (User) bzw. Akteur und dem System. Ein Anwednungsfall geht nicht auf die Details der Benutzerschnittstelle ein, ausser sie wären zentral für das Systemverhalten.

4.1.1 Neues Spiel beginnen

Akteur	Spieler
Kurzbeschreibung	Ein neues Spiel beginnen
Vorbedingungen	Keine
Nachzustand	Geheimcode ist zufällig gesetzt Spielbrett ist leer
Fehlersituationen	Keine
Nachzustände im Fehlerfall	Keine
Standardablauf	Spieler wählt "neues Spiel" System macht Rückfrage System initialisiert Spiel
Alternativabläufe	Wird automatisch nach dem Programmstart ausgeführt
Regeln	Keine

4.1.2 Codewort eingeben

Akteur	Spieler
Kurzbeschreibung	Ein Codewort zum Erraten des Codes eingeben
Vorbedingungen	Spiel gestartet
Nachzustand	Codewort eingegeben
Fehlersituationen	In falscher Zeile eingegeben (z.B. in leerer oder in schon korrigierter Zeile)
Nachzustände im Fehlerfall	Aufforderung zur Eingabe in richtiger Zeile
Standardablauf	Spieler wählt für jedes der vier Felder eine Farbe
Alternativabläufe	keine
Regeln	Jede Farbe nur ein Mal benutzen

4.1.3 Code auswerten

Akteur	System
Kurzbeschreibung	Code wird vom System ausgewertet
Vorbedingungen	Code eingegeben
Nachzustand	Korrektur gemacht, bewertet
Fehlersituationen	Nicht alle vier Felder gesetzt
Nachzustände im Fehlerfall	Aufforderung zum Ausfüllen aller Felder
Standardablauf	1. System vergleicht Codewort mit Lösung 2. System wertet aus 3. System gibt Resultat auf GUI aus
Alternativabläufe	keine
Regeln	Weiss: Farbe richtig, aber am falschen Ort Schwarz: Farbe und Ort richtig Die Reihenfolge der Lösungsstecker hängt nicht mit der Reihenfolge der Lösung zusammen

4.1.4 Code aufdecken

Akteur	Spieler
Kurzbeschreibung	Spieler gibt auf und möchte die Lösung sehen
Vorbedingungen	Spiel gestartet
Nachzustand	Code (Lösung) ist aufgedeckt
Fehlersituationen	keine
Nachzustände im Fehlerfall	keine
Standardablauf	1. Spieler drückt "Spiel beenden" 2. System deckt Lösung auf
Alternativabläufe	keine
Regeln	keine

4.1.5 Spielende

Akteur	System
Kurzbeschreibung	Spiel ist zu Ende weil Code erraten oder weil Spieler verloren (nicht geschafft in 8 Versuchen)
Vorbedingungen	- 8 erfolglose Versuche - Spieler hat Code erraten
Nachzustand	- GUI-Ausgabe: Gratulation - Highscore zeigen und speichern
Fehlersituationen	keine
Nachzustände im Fehlerfall	keine
Standardablauf	1. Spieler hat nicht gewonnen 2. GUI-Ausgabe: Pech gehabt 3. Lösung aufdecken (Siehe Anwendungsfall "Code aufdecken") 4. Highscore anzeigen
Alternativabläufe	1. Spieler hat Code in weniger als 8 Versuchen erraten 2. Name vom Spieler abfragen 3. Eintrag in Highscoreliste 4. Highscore anzeigen
Regeln	Ein Spiel ist beendet falls Code erraten, oder falls Code nicht in weniger als 8 Versuchen erraten.

4.1.6 Spiel beenden

Akteur	Spieler
Kurzbeschreibung	Applikation wird beendet
Vorbedingungen	Applikation läuft
Nachzustand	Applikation ist beendet
Fehlersituationen	keine
Nachzustände im Fehlerfall	keine
Standardablauf	1. Spieler drückt "Beenden" 2. System beendet
Alternativabläufe	keine
Regeln	keine

4.2 GUI-Design

Folgende Bilder zeigen die ersten Entwürfe unserer Oberfläche. Die Designvorschläge beinhalten alle definierten Elemente. Die Anordnung und definitive Implementierung wird sich dann beim Programmieren des Graphical User Interfaces ergeben.

4.2.1 Hauptformular

Das Hauptformular besteht im wesentlichen aus dem Spielbrett (Matrix) und dem Pin-Vorrat (Abbildung 4.1). Über der Spielmatrix wird die Lösung im Falle, dass das Spiel gelöst wurde angezeigt. Ebenfalls soll der Username während dem Spiel am oberen Rand dargestellt werden.

4.2.2 Dialoge

Dialoge gibt es im Wesentlichen in unserem Mastermind nur einen. Der Optionsdialog (Abbildung 4.2) bietet dem Benutzer die Möglichkeit die Spieleinstellungen anzupassen. In ihm werden beispielsweise der Playername, die Anzahl Pins und die Grösse der Matrix, das Aussehen der Pins, usw. festgelegt.

4.2.3 Menüs

Jede Aktion kann über einen Menüpunkt aufgerufen werden. Die Menüs (Abbildung 4.3) sind in die Kategorien Datei, Spiel und Hilfe gegliedert. Unter "Datei" sind alle Punkte für die grobe Steuerung (Speichern, Beenden, usw.)

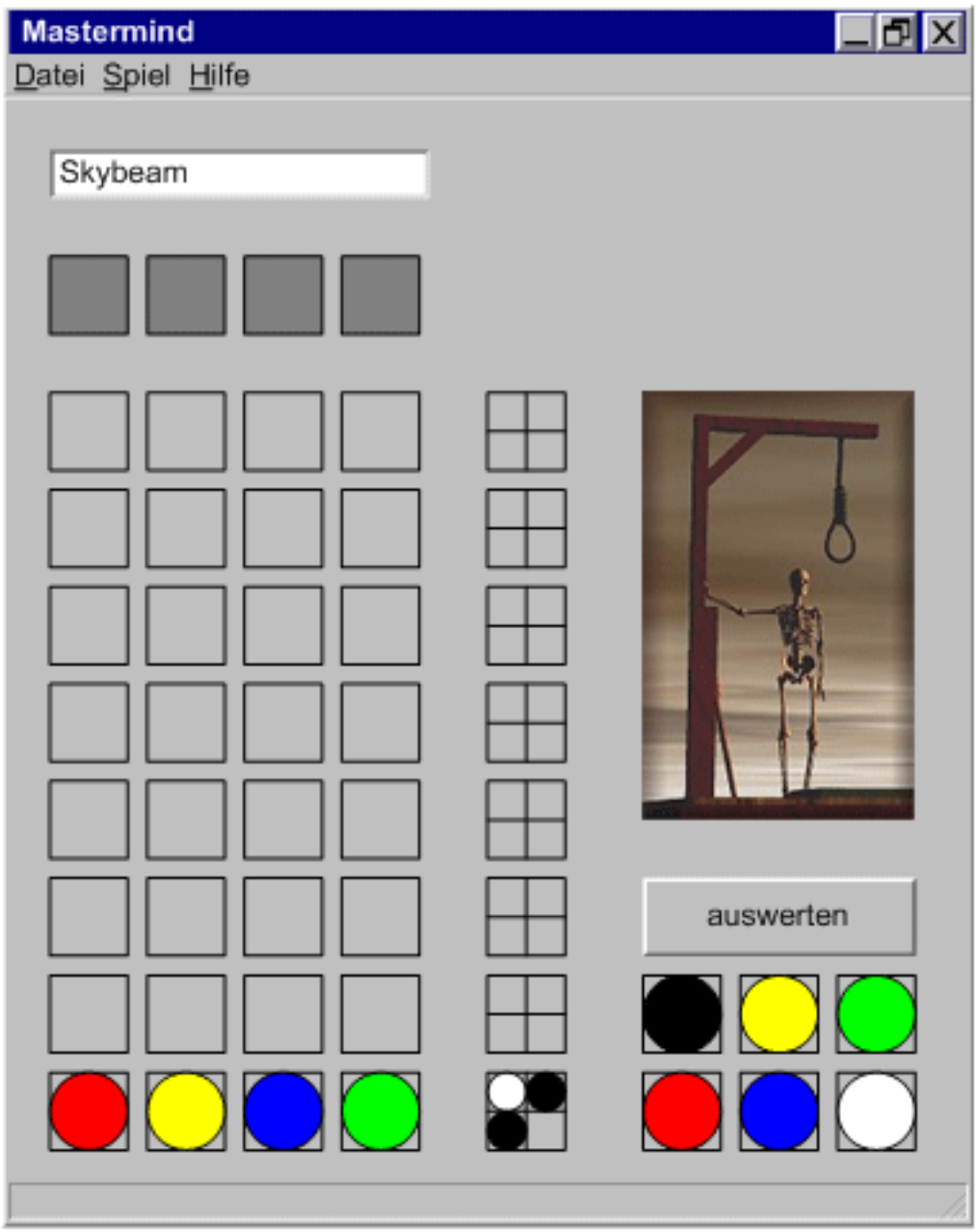

Abbildung 4.1: das Hauptformular der Applikation

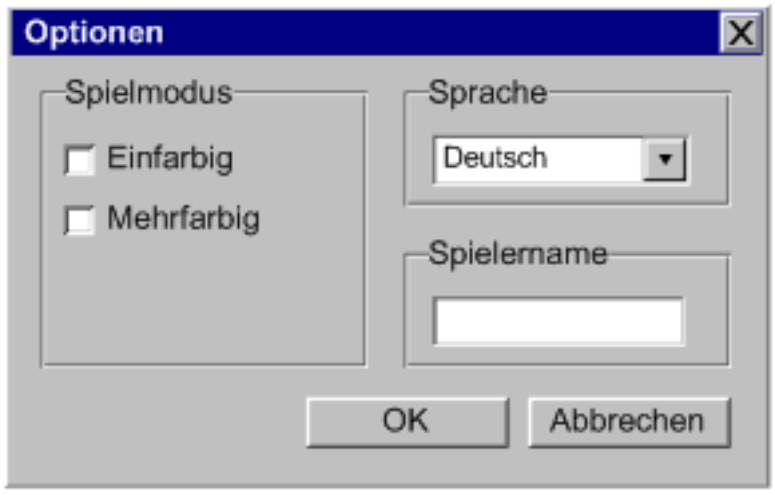

Abbildung 4.2: der Optionsdialog der Applikation

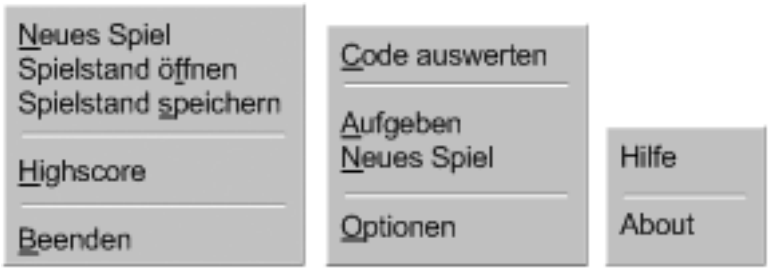

Abbildung 4.3: die Menüs "Datei", "Spiel" und "Hilfe" der Applikation

zu finden. "Spiel" umfasst alle relevanten Menüeinträge um ein Spiel zu spielen. Über den Menüpunkt Hilfe gelangt man zu der eigentlichen Online-Hilfe und der About-Box.

Kapitel 5

Realisierung

In der ganze Realisierungsphase wurde sehr viel Wert auf das MVC-Konzept gelegt (Abbildung 5.1). Aus diesem Grund sind auch unsere 3 Hauptpackages Model, View und Control entstanden. Wir haben das grundlegende MVC-Konzept ein wenig auf unsere Bedürfnisse angepasst. In unserer Version fiel der Update-Link vom Model zur View weg, da jegliche Kommunikation zwischen dem Model und der View über den Controller abgewickelt wird.

5.1 UML Diagramme

5.1.1 Klassendiagramme

In unserem UML-Klassendiagramm erkennt man anhand der Packagedefinitionen die Implementation des MVC-Konzeptes. In der Planungsphase wurde der grösste Teil unserer UML-Diagramme bereits erstellt. Einzig die extrem spezifischen GUI-Klassen wurden nachträglich während der Realisierung ein-

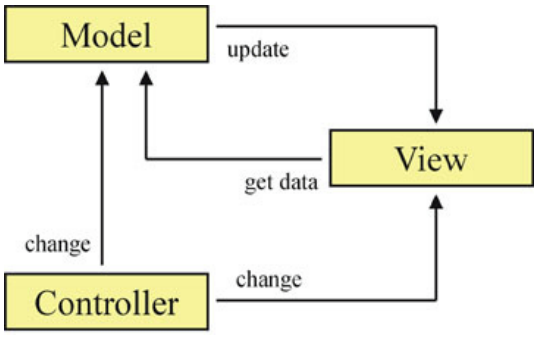

Abbildung 5.1: das grundlegende MVC-Konzept

gefügt, da während der Planung nur die Haupt-GUI-Klasse zur Diskussion stand.
Unser UML-Diagramm basiert auf einem Hauptpackage mastermind und 3 Unterpackages model, view und control. Im Model-Package befinden sich alle Klassen welche in irgend einer Weise dazu dienen die Daten zu speichern. Das Control-Package beinhaltet genau eine Klasse, die Contoller-Klasse. Alle Klassen welche irgend eine darstellungstechnische Funktion wahrnehmen befinden sich im View-Package.
Für eine detaillierte Funktionsbeschreibung der einzelnen Klassen und deren Methoden konsultieren Sie bitte unsere beiliegende JavaDoc.

5.1.2 Sequenzdiagramme

Ebenfalls wurden bereits während der Analyse- und Designphase, nachdem das UML-Klassendiagramm definiert war, einige Sequenzdiagramme angefertigt. Diese UML-Sequenzdiagramme wiederspiegeln nicht mehr genau den Ist-Zustand, aber die eigentliche Kommunikation, bzw. die Interaktionen zwischen den einzelnen MVC-Elementen wird immer noch schön visualisiert.
Als Beispiel dient die Aktion "Spiel Starten" (Abbildung 5.8). An diesem Beispiel kann man einfach nachvollziehen, wie der Benutzer auf dem GUI (durch eine Aktion) den Event an den Controller auslöst. Dementsprechend erstellt der Controller anschliessen den ratenden Spieler (in dem Fall das Objekt für den menschlichen Spieler mit Username, usw.) und das Objekt für den vorgebenden Spieler (in diesem Fall der Computer). Anschliessend erstellt er mit den neuen Werten eine neue Matrix und registriert sich auch gleich bei ihr als Listener. Anschliessend benachrichtigt er das GUI, dass sich die Matrix geändert habe und veranlasst so einen Repaint des Spielfeldes.
Ähnlich laufen die Aktionen beim Setzen von einem neuen Pin ab (Abbildung 5.9). Wiederum geht die Aktion vom Benutzer aus, welcher auf dem GUI einen Pin auf die Spielfläche gesetzt hat. Das GUI sendet wieder einen Event (mit den Pindaten) ab. Der Controller als Listener für diesen Event reagiert entsprechend und veranlasst die Matrix diesen Spielzug zu speichern. Die Matrix wiederum feuert den nächsten Event, dass Sie sich geändert habe und das GUI zeichnet darauf ihr Spielfeld neu. So wird verhindert, dass zwischen GUI-Spielfeld und den Daten in der Matrix Unterschiede entstehen. Auch wenn etwas anderes unsere Matrix verändern würde, so würde sich unser GUI automatisch aktualisieren, da die Matrix bei jeder Veränderung einen Event feuert.

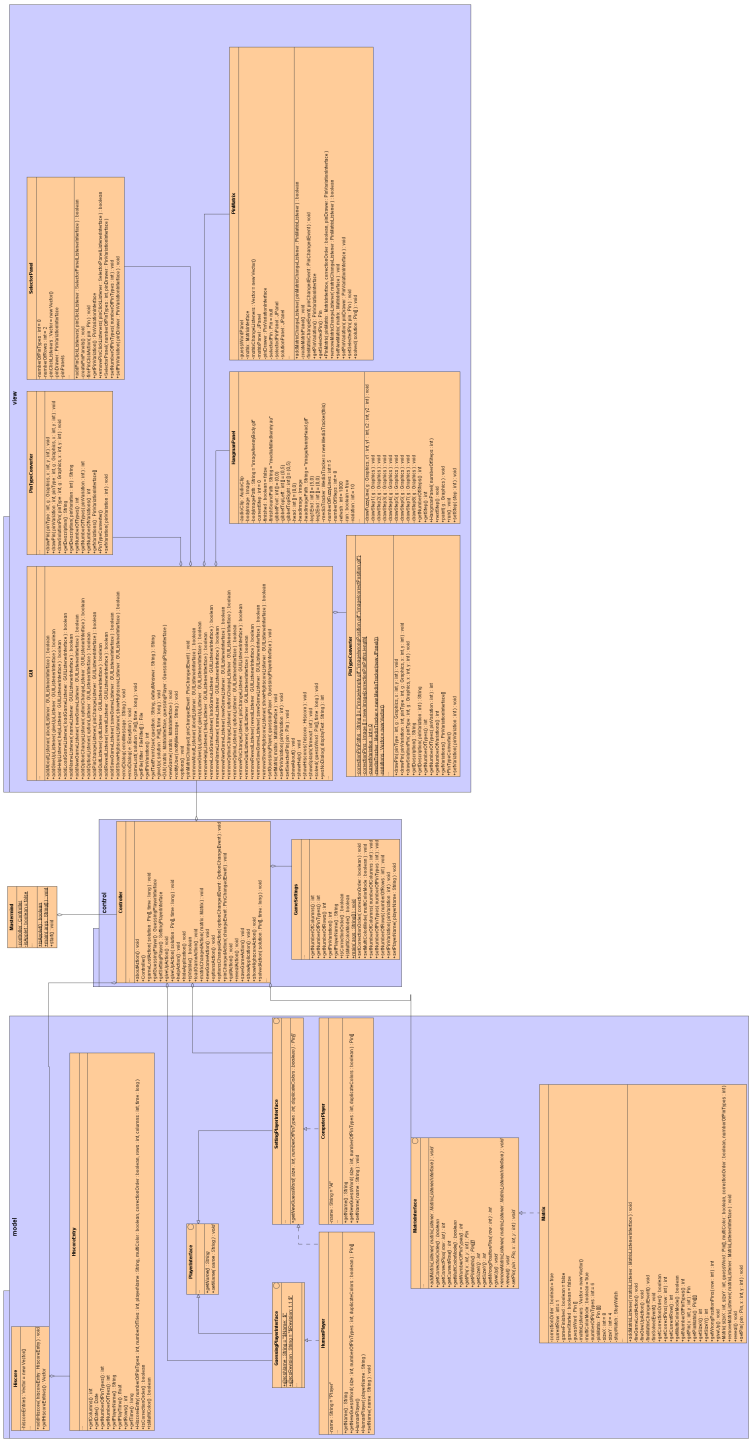

Abbildung 5.2: unser UML-Diagramm

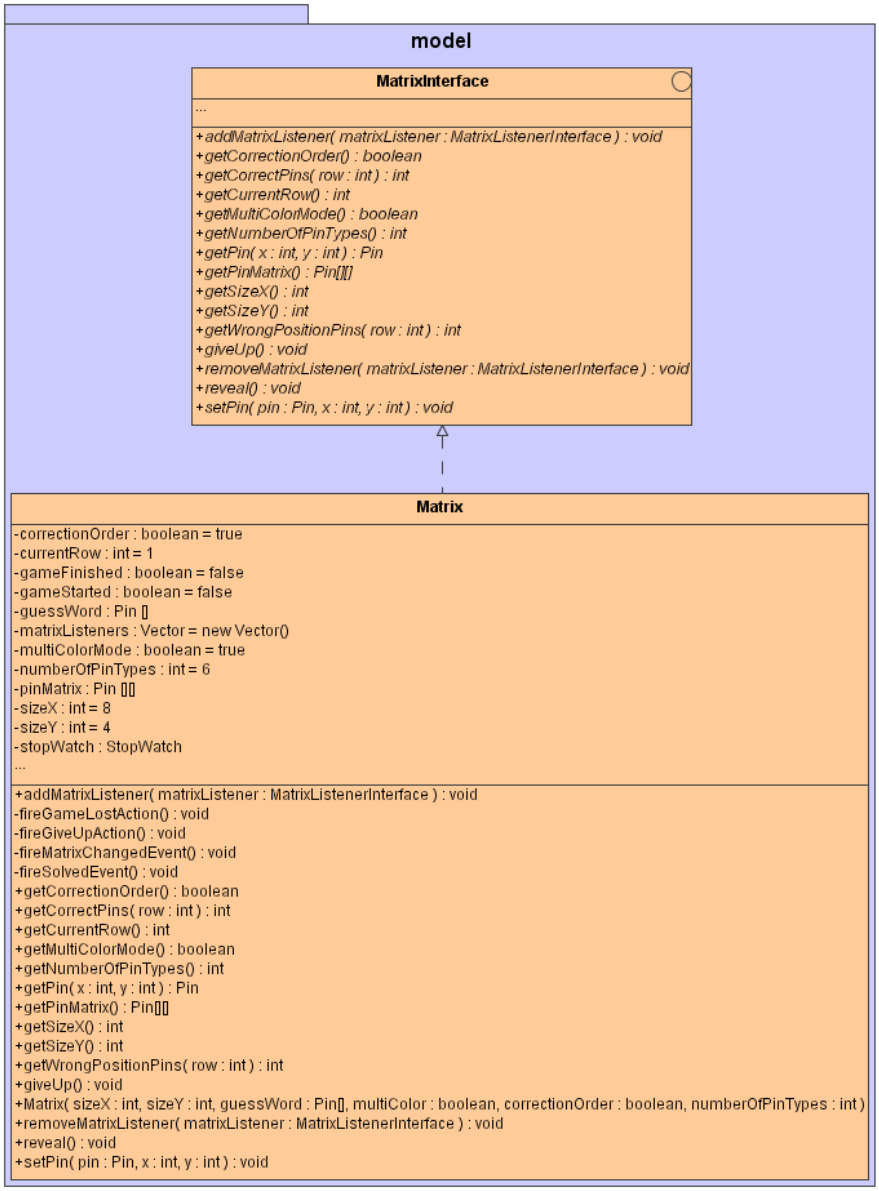

Abbildung 5.3: das Matrix-Model

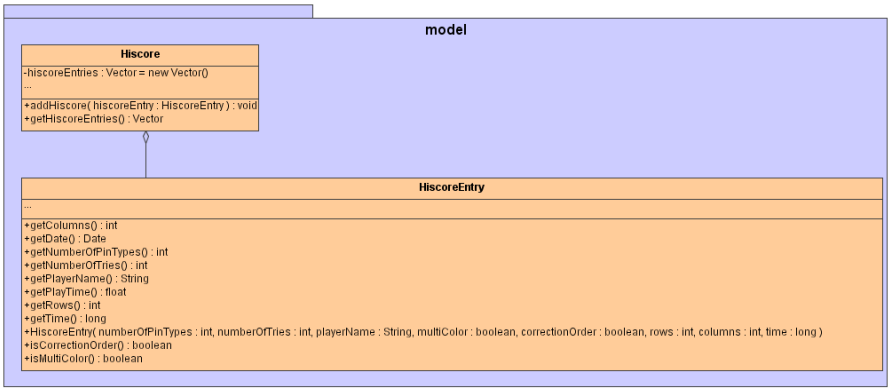

Abbildung 5.4: das Hiscore-Model

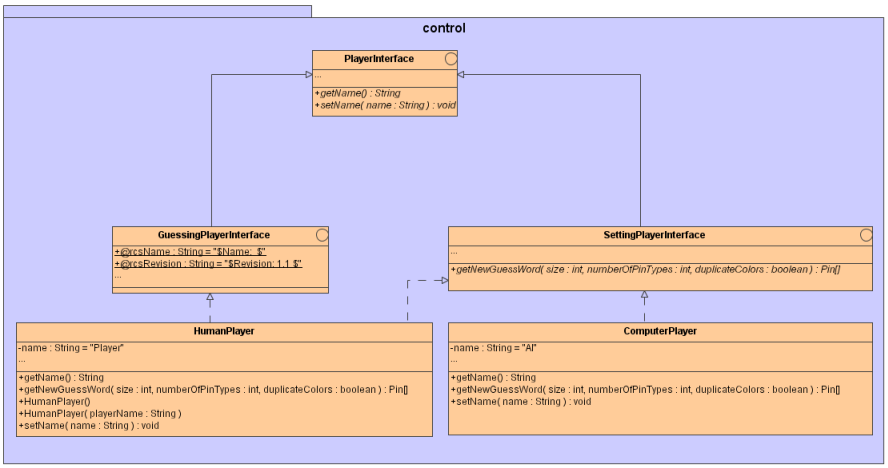

Abbildung 5.5: das Player-Model

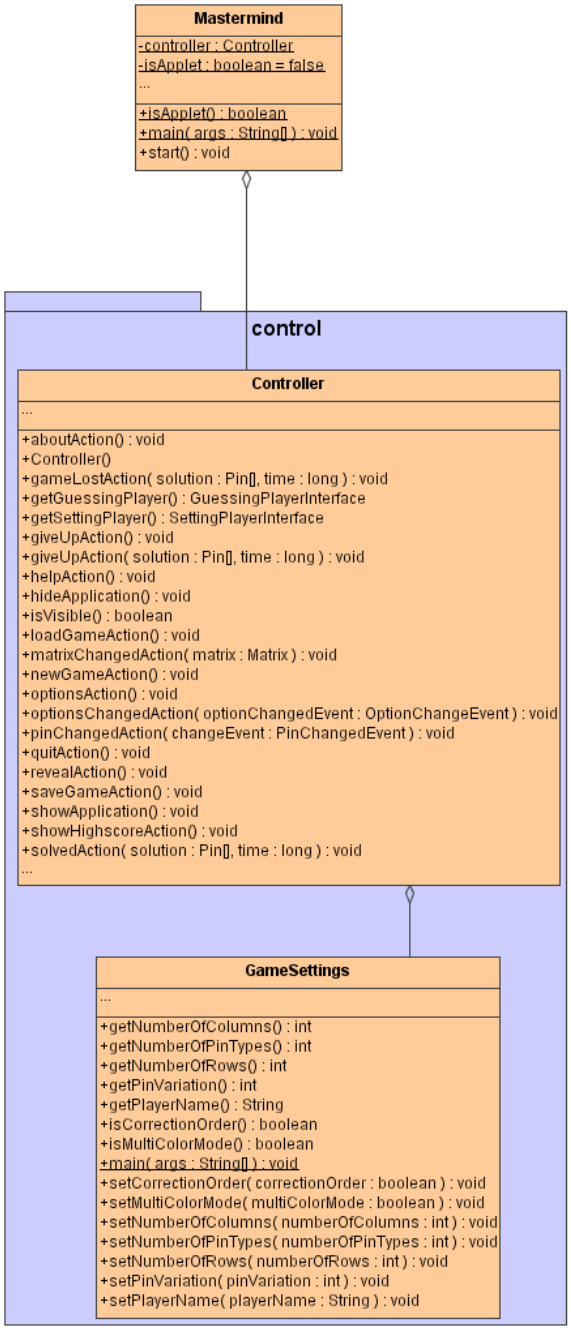

Abbildung 5.6: das Contol-Package

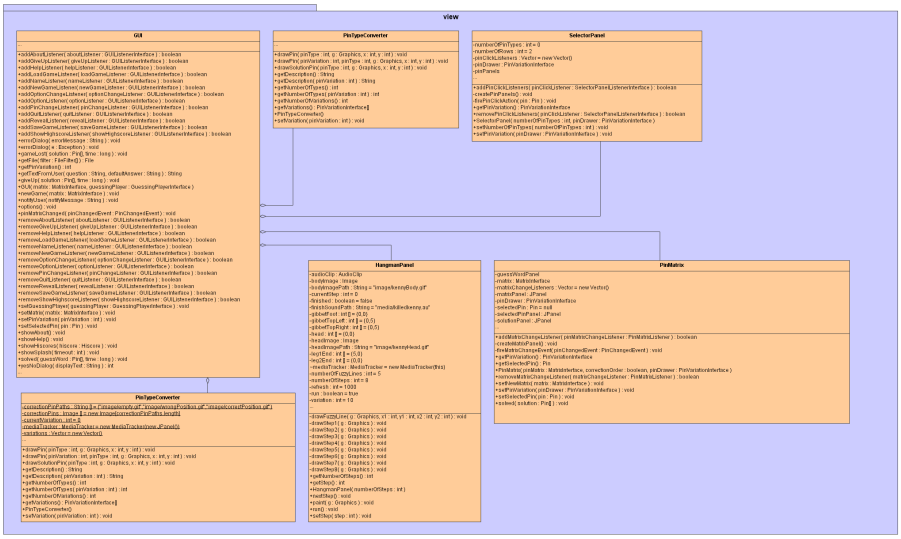

Abbildung 5.7: das View-Package

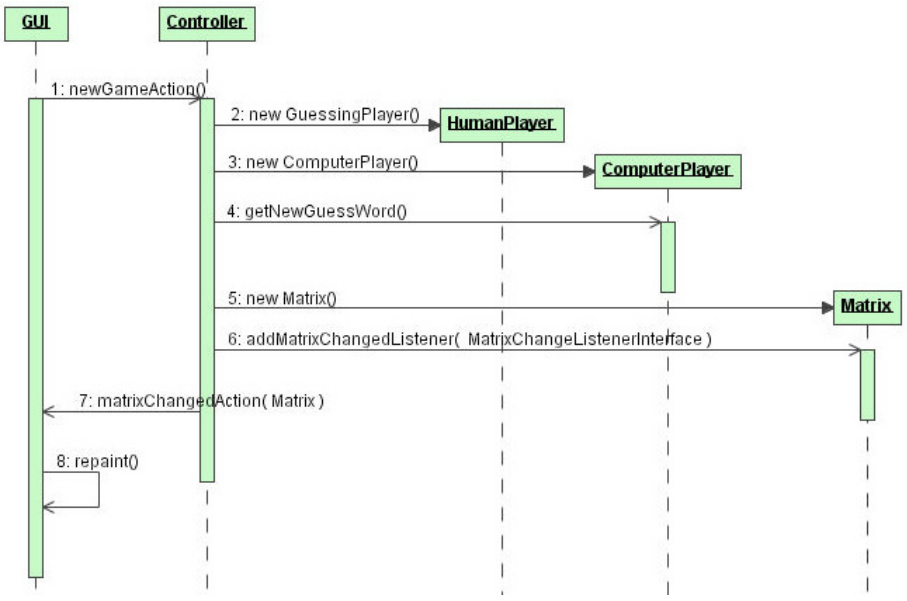

Abbildung 5.8: UML-Sequenzdiagramm "Spiel starten"

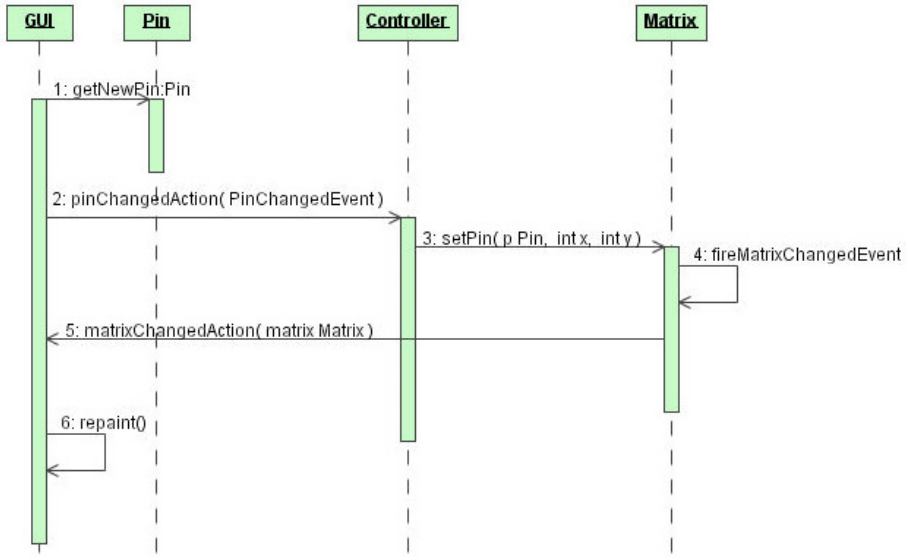

Abbildung 5.9: UML-Sequenzdiagramm "Pin setzen"

5.2 Screenshots

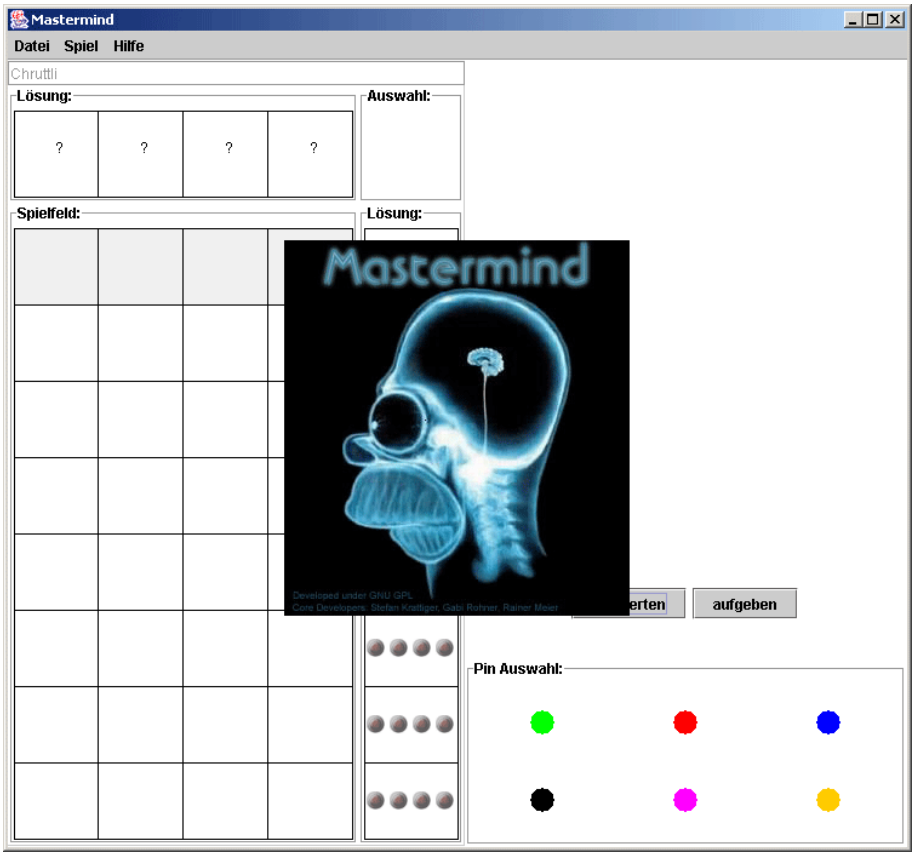

Abbildung 5.10: der Splash-Screen

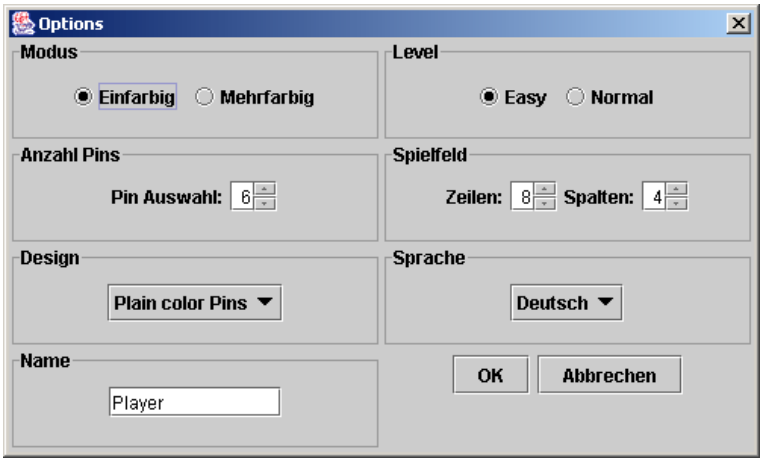

Abbildung 5.11: der Optionsdialog

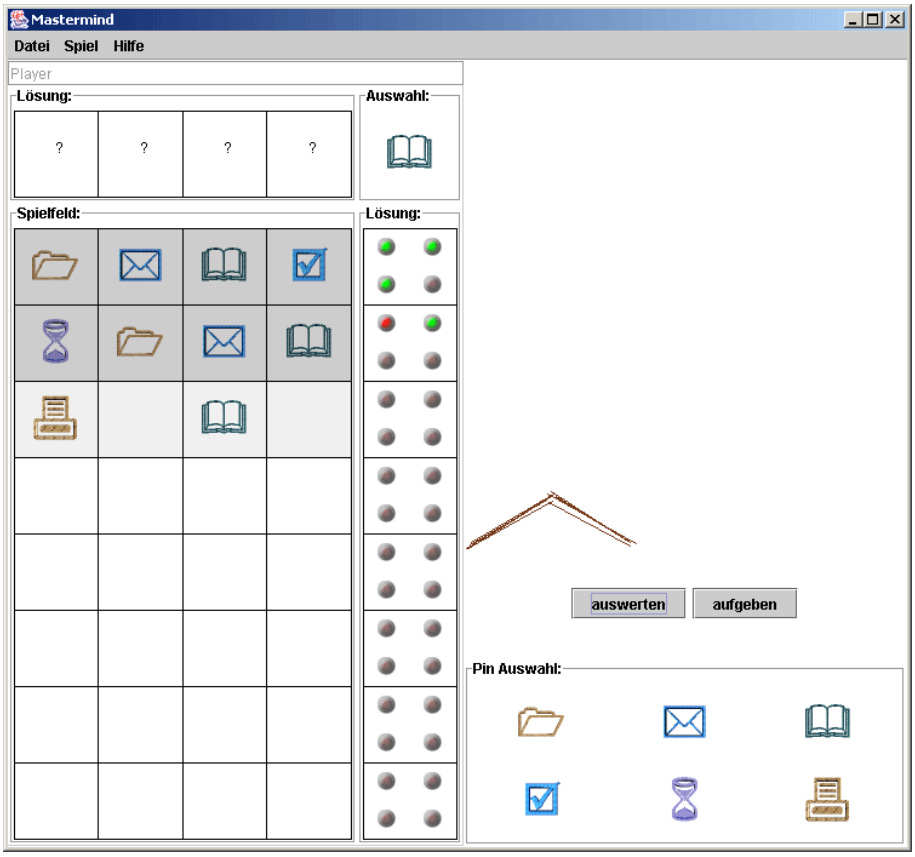

Abbildung 5.12: das Spielfeld

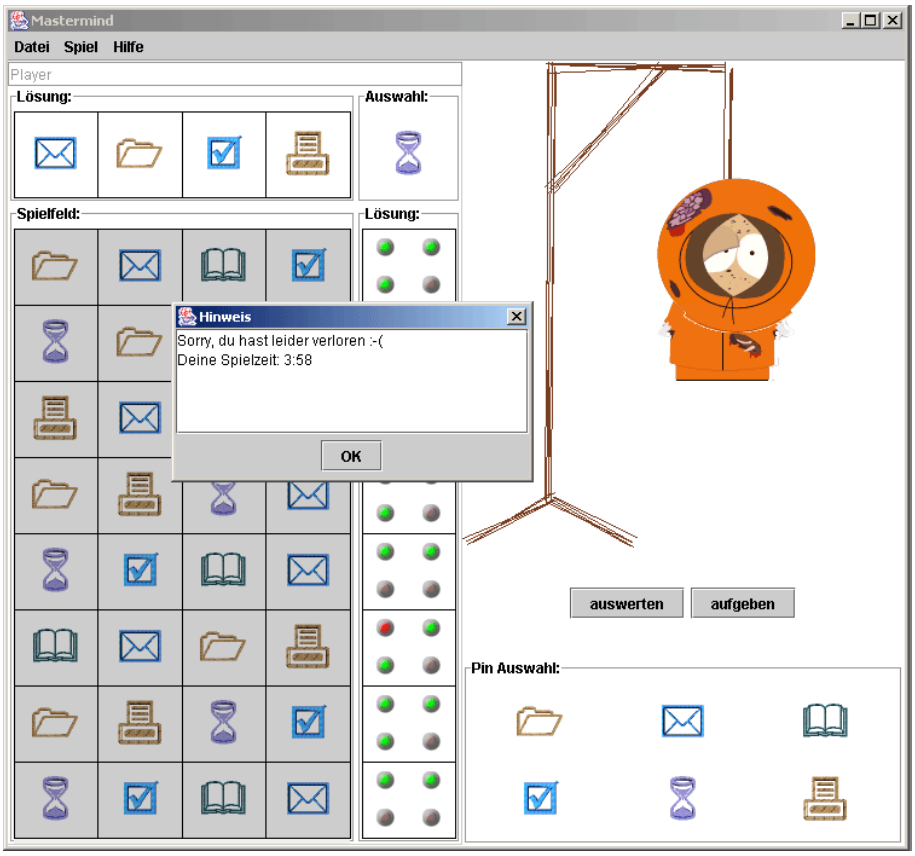

Abbildung 5.13: ein verlorenes Game mit gehängtem Kenny

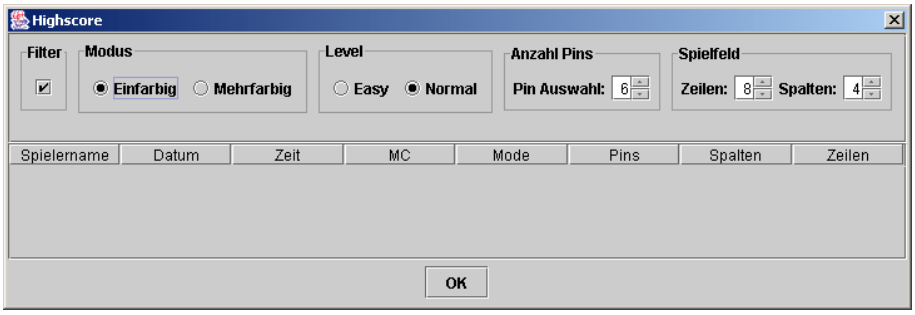

Abbildung 5.14: die Hiscore-Tabelle

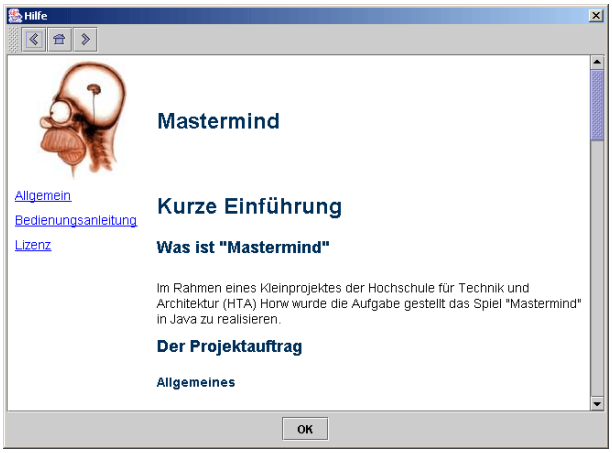

Abbildung 5.15: die Hilfe

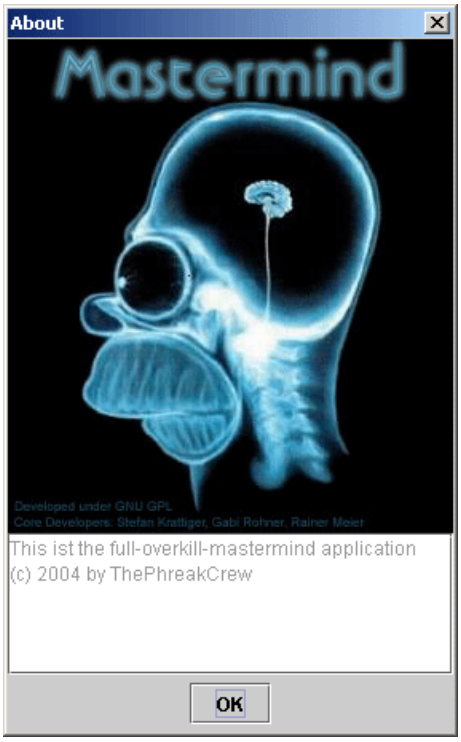

Abbildung 5.16: der About-Dialog

5.3 Zusätzliche Features

Während der ganzen Realisierungsphase kamen uns immer wieder irgendwelche speziellen Sonderfunktionen in den Sinn. Diese Sonderfunktionen hatten nicht immer einen direkten Zusammenhang mit dem Mastermind-Spiel, vielmehr waren es Funktionen, von denen man einmal gehört hatte und gerne einmal programmiert hätte. Diese Wünsche wurden aufgenommen und ohne Priorität teilweise entwickelt. So entstanden diverse Funktionen, hinter denen eine grosse Menge Arbeit und Spass stehen und hier dokumentiert werden.

5.3.1 Hangman

Die Idee des mitwachsenden Hangmans ist eher zufällig entstanden, nachdem ein Projektteammitglied diese beiden Spiele einmal namentlich vertauscht hatte. Die Idee wurde analysiert und in Angriff genommen. Der Hangman (Kenny) wächst dynamisch mit jeder Linie die man im Mastermind nicht erraten hat. Wurde das Spiel nicht gelöst, so "hängt" Kenny und der typische Sound "Oh my god, they killed Kenny!" erklingt aus den PC-Speakers. Wie anfänglich erwähnt, hat jedes Feature auch einen gewissen Programmierreiz hervorgerufen. Deshalb wurde das Zeichnen des Hangmans auch mit einem Thread realisiert, welcher die Linien zusätzlich animiert.

5.3.2 Konfiguration

Die "normale" Ausführung eines Masterminds besteht aus einem Spielfeld mit 4 Spalten, 8 Reihen und 6 unterschiedliche Pins. Unser Mastermind kann individuell konfiguriert werden. Das heisst, jegliche Spielgrössen können angepasst werden. Beispielsweise kann die Spielfeldgrösse grösser oder kleiner eingestellt werden. Wem die Standardpins zu langweilig sind kann auch zwischen diversen anderen Variationen eine auswählen.

5.3.3 Öffnen / Speichern

Anfänglich war auch angedacht, dass man zu einem beliebigen Zeitpunkt sein begonnenes Spiel zu speichern und zu einem späteren Zeitpunkt fortsetzen. Der Grundgedanke hinter dieser Idee war das Interesse an der Serialisierung von Objekten. Die ganze Geschichte wurde dann auch implementiert und ausführlich getestet. Einige Zeit später, nachdem unser Mastermind einige realen Test unterzogen wurde, haben wir uns entschieden die beiden Funktionen zum Öffnen und Speichern der Spielstände wieder zu entfernen. Diese Funktionen hatten einen grossen Nachteil: Wird ein Spiel gestartet und dann

bereits gespeichert, so kann man in aller Ruhe das Spiel lösen. Mit der so erratenen Lösung kann man nun den alten Spielstand wieder laden und das Spiel nach einem Spielzug und wenigen Sekunden Zeit lösen. Diese Funktionen dienten also dem Verfälschen unserer Hiscore und genau das wollten wir von Anfang an verhindern. Deshalb können in der aktuellen Version die Spielstände nicht mehr gespeichert, bzw. geladen werden.

5.3.4 Applet / Application

Unser Mastermind kann wahlweise als Java Application oder als Java Applet ausgeführt werden. Der Clou daran ist jedoch, dass nur ein Java Archive File existiert und die beiden Varianten in der selben Code-Struktur realisiert wurden.

Applet-Code

```
 <applet code=''ch.skybeam.mastermind.Mastermind''
archive=''Mastermind.jar'' name=''Mastermind'' width = ''686''
height=''385'' id=''Mastermind''> </applet>
```

5.3.5 Debuglevels

Während der ganzen Entwicklung mussten immer wieder diverse Debug-Ausgaben geschrieben werden. Diese ganzen Meldungen wurden über eine zentrale Funktion ausgegeben. Die Debug-Meldungen wurden in verschiedene Levels unterteilt:

Level 1 Normale Debug-Meldungen

Level 9 Debug-Meldungen mit zusätzlichen Thread-Informationen

Die Debug-Ausgaben können mit folgendem Befehl aktiviert werden:
`java -DDEBUG=[Debuglevel] -jar Mastermind.jar`

Kapitel 6

Testing

Mastermind wurde in zwei Testbereiche unterteilt. Das funktionale Testing wurde mittels JUnit-Testcases realisiert. Für den zweiten Bereich, dem User-Testing, wurden verschiedene Test-Szenarios aufgestellt und geprüft ob die Applikation auch entsprechend unseren Erwartungen reagiert.

6.1 User-Testing

Die verschiedenen Testfälle wurden (mit Hilfe der anfänglich definierten Use-Cases) festgelegt und anschliessend durch Benutzer durchgeführt.

6.1.1 Die Testfälle

Nr.	Beschreibung	Erwartetes Resultat	Ergebnis
1	Reihe auswerten bevor die ganze Reihe gefüllt ist	Meldung, dass zuerst die ganze Reihe ausgefüllt werden muss	Dialog: "Sie haben versucht eine unvollständige Reihe auszuwerten. Bitte vervollständigen Sie zuerst die Reihe!"
2	Reihe falsch ausfüllen	Hangman baut sich um eine Stufe auf.	Hangman baut sich auf
3	Codewort in "Anzahl Zeilen" - Zügen nicht erraten.	Kenny ist tot / gehängt.	Hangman wurde in den Anzahl Zeilen vollständig aufgebaut.

4	Optionen einstellen auf: Modus = Mehrfarbig, Level = Normal, Anzahl Pins = 8, Spielfeld: 8x4, Design = numbers, Name = Gabi	Mehrfachfarben erlaubt, Korrektur zeigt nicht an welcher Punkt richtig war, Spielfeld ist 8x4, Design ist Zahlen, Spielername ist Gabi	Meldung, dass Optionen verändert und deshalb ein neues Spiel gestartet werden muss. Bestätigung durch OK.
5	Spiel ist gewonnen.	Lösung wird aufgedeckt.	Lösung aufgedeckt
6	Spiel wird aufgegeben, indem der Button "aufgeben" gedrückt wird.	Lösung wird aufgedeckt.	In den Lösungsfeldern erscheinen die Lösungspins
7	Pin in der Pinauswahl anklicken.	Im Feld "Auswahl" erscheint der gewählte Pin.	Nachdem der Pin mit der Maus gewählt wurde, wird er in der Auswahl angezeigt
8	In den Optionen den Spielernamen angeben.	Name erscheint auf dem Spielbrett.	Korrekter Name ist auf dem Spielbrett sichtbar
9	Hilfe anzeigen	Beschreibung des Mastermindes wird angezeigt.	Dialog mit Benutzeranleitung wird geöffnet
10	Spiel beenden	Mastermind wird geschlossen Fenster	Mastermind wird geschlossen
11	Spiel "neues Spiel"	Spielfeld wird geleert.	Neues Spielfeld mit neuem Codewort zum erraten gezeichnet
12	Hiscore öffnen	Hiscore wird angezeigt	Dialog mit Highscore öffnet sich
13	Zeilen ausfüllen und überprüfen, ob richtig korrigiert wird.	Korrekturpunkte richtig anhand des Levels und der Übereinstimmungen mit der Lösung	Nach mehrmaligen Tests kann man sagen, dass die Korrektur richtig funktioniert

Kapitel 7

Quellen / Tools

http://www.skybeam.ch/mastermind Unsere Projektwebseite

http://www.adarvo.ch Jegliche Dokumente (ohne Quellcode) wurden über Adarvo ThemeWare ausgetauscht / verwaltet.

http://www.cvshome.org Unser gesamter Sourcecode wurde mittels CVS-System verwaltet.

http://www.eclipse.org Als IDE für Java setzten wir Eclipse ein.

http://java.com/de Software Development Kit for Java

http://www.magicdraw.com Der von uns eingesetzte UML-Designer

http://www.dante.de Dieses Dokument wurde mit LaTeX 2_ε geschrieben

http://www.uml.org Alle Code-Visualisierungen wurden mit der Unified Modelling Language erstellt

Abbildungsverzeichnis