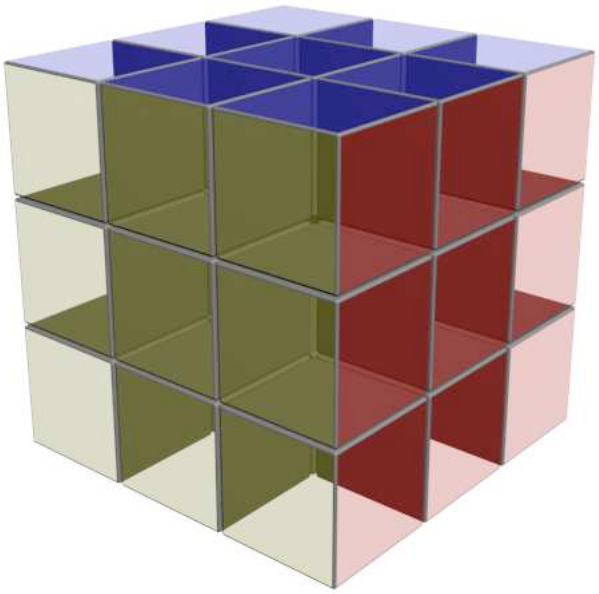
UNIVERSITÄT PASSAU

Software Engineering Praktikum WS 06/07
Gruppe 2

»Implementierung« v 1.0

YaRC

Yet another Rubik Cube



Phase	Phasenverantwortliche(r)	E-Mail
[redacted]		

Inhaltsverzeichnis

1 Einleitung

In diesem Dokument wird die Implementierungsphase des SEP-Projekts *Rubikwürfel auf J2ME-Handy* beschrieben. Dabei wir zuerst auf die Probleme und Änderungen im Vergleich zum Entwurf bzw. der Feinspezifikation eingegangen.
Der nächste Abschnitt behandelt dann die Testfälle (J2Unit-Tests), mit deren Hilfe die Korrektheit der Implementierung der einzelnen Klassen getestet wurde. Es handelt sich dabei um statistische Auswertungen der Testergebnisse.
Im dritten Teil wird der Implementierungsplan aus der Feinspezifikationsphase mit der Realität verglichen.
Im Anhang dieses Dokuments befindet sich das Benutzerhandbuch zu diesem Handyspiel.

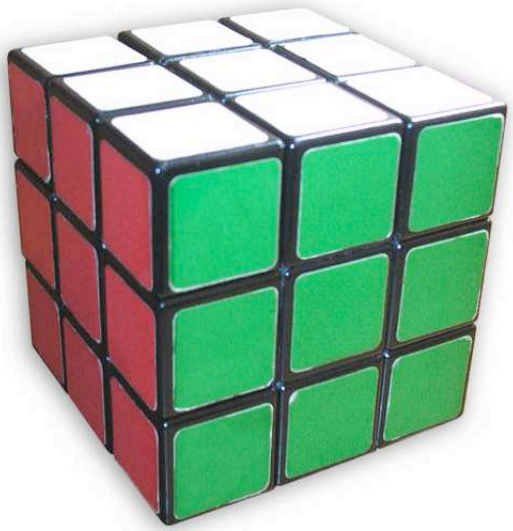


Abbildung 1 - Rubik-Würfel

2 Probleme und Änderungen am Entwurf

In diesem Kapitel werden die Änderungen, die sich während der Implementierung ergeben haben dokumentiert. Die Klassen sind dabei in alphabetischer Reihenfolge angeführt.

2.1 Model

2.1.1 BestenlisteBerechnen

Keine Änderungen.

2.1.2 Drehung

Keine Änderungen.

2.1.3 EckWuerfel

Keine Änderungen.

2.1.4 EffizienterLoeser

Die Klasse wurde ohne Änderungen implementiert, jedoch wird sie nicht mehr verwendet. Es war vorgessen, dass der EffizienterLoeser bei einem RubikWuerfel im Bildschirmschoner-Modus zum Einsatz kommt. Hier wird aber jetzt die Klasse IntuitiverLoser eingesetzt, da es für den Benutzer anschaulicher ist.

2.1.5 Einstellungen

Diese Klasse wurde neu angelegt, um die Einstellungen zu kapseln.

2.1.6 IntuitiverLoeser

Folgende Methoden sind nicht mehr vorhanden:

- bei Ebene2ansetzen() Wurde entfernt, weil die Hilfefunktion gleich die ganze Ebene 2 löst.
- loeseEbene3() Wurde entfernt, da die Hilfefunktion immer nur einen Finalzug ausführt.

Folgende Methode ist neu hinzu gekommen:

- finalzugAusfuehren() Weil die Hilfefunktion immer nur einen Finalzug ausführt.

2.1.7 KantenWuerfel

Keine Änderungen.

2.1.8 ListenEintrag

Keine Änderungen.

2.1.9 LoeserInterface

Keine Änderungen.

2.1.10 MittelWuerfel

Keine Änderungen.

2.1.11 Position

Keine Änderungen.

2.1.12 RubikWuerfel

- Die Methode public boolean eckUndKantenWuerfelRichtig( ) ist nicht mehr vorhanden, da wenn die Hilfefunktion aufgerufen wird und es ist noch keine Reihe gelöst, einfach die Reihe 1 gelöst wird. Somit wird nicht im RubikWuerfel geschaut, ob ein Eck- und KantenWuerfel bereits in irgendeiner Reihe richtig liegen.
- Diese Methode wurde hinzugefügt: private int[] anzahlRichtigerTeilwuerfel(int reihe). Diese Methode ist nur eine Hilfsmethode für die Methode sucheAmMeistenGeloesteReihe. Durch diese Aufteilung wird die Klasse RubikWuerfel besser strukturiert.
- Bei der Methode public Vector setzeGeloesteReiheOben( ) wurde die Signatur verändert, ursprünglich: public Vector setzeGeloesteReiheOben(boolean reiheGeloest). Grund: Da wenn die Hilfefunktion aufgerufen wird und es ist noch keine Reihe gelöst, einfach die Reihe 1 gelöst wird. Somit wird nicht im RubikWuerfel geschaut, ob ein Eck- und KantenWuerfel bereits in irgendeiner Reihe richtig liegen, und damit muss in diesem Fall keine Reihe nach oben gesetzt werden. -> Die Unterscheidung zwischen Reihe gelöst und Reihe nicht gelöst fällt somit weg, da nur noch im Fall Reihe gelöst diese Methode aufgerufen wird.

2.1.13 SpielSpeicher

- Es ist das private-Attribut boolean oberflaeche hinzugekommen. Grund: Da man wissen muss, ob bei dem gespeicherten Spiel der RubikWuerfel ein Texturenwuerfel oder ein Farbwuerfel ist.
- Aus den vorherigen Punkt ergibt sich, dass die Methode public boolean getOberflaeche( ) hinzugefügt wurde.
- Die Methode public String toString( ) wurde zur Kontrolle hinzugefügt.

2.1.14 Startgenerator

Keine Änderungen.

2.1.15 Teilwuerfel

Keine Änderungen.

2.2 Controller

2.2.1 BestenlisteManager

- private final int LAENGE wurde in MAXIMALE_LAENGE umbenannt
- private boolean datenGeaendert hinzugefügt. In diesem Attribut wird gespeichert, ob die Daten sich geandert haben.
- public String getHighscore() wurde in public ListenEintrag[] getBestenliste() umbenannt. Die Methode liefert die Bestenliste als Array zurück
- public static BestenlisteManager gibInstanz() hinzugefügt; gibt eine Instanz des BestenlisteManagers zurück.
- private void ladeBestenliste() hinzugefügt; liefert die Bestenliste als Array zurück
- public boolean[] getHighscorePlatz(int schwierigkeitsgrad,long gespielteZeit) hinzugefügt; ermittelt den HighscorePlatz, wobei schwierigkeitsgrad der Schwierigkeitsgrad des soeben bendeten Spieles ist, und gespielteZeit wie lange das letze Spiel gedauert hat.

2.2.2 EinstellungenManager

- private boolean einstellungenGeaendert() hinzugefügt; die Methode steuert das Verhalten der laden()-Methode.Die Einstellungen werden nur dann neu geladen, wenn sie (durch einen Aufruf von speichern()) geändert wurden.
- private int sprache, private boolean ton, private boolean oberflaeche, private int lautstaerke, private int schwierigkeitsgrad, public static final boolean TON_AN, public static final boolean TON_AUS, public static final boolean TEXTUR, public static final boolean FARBE, public static final int PROFI, public static final int ANFAENGER, public static final int FORTGESCHRITTEN, public static final int UNVERDREHT wurden vom EinstellungenManager entfernt. Siehe dazu die Klasse Einstellungen im Model.
- public void loescheEinstellungen() hinzugefügt; die Methode löscht alle gespeicherten Einstellungen.
- private void speichern() hinzugefügt; die Methode speichert die Einstellungen des Spieles.
- private void laden() hinzugefügt; die Methode lädt die gespeicherten Einstellungen.

2.2.3 HistorieManager

- HistorieManager() Es war kein Konstruktor definiert.
- public boolean istUndoMoeglich() Ist noch ein Element auf dem Stack oder nicht.
- public boolean istRedoMoeglich() Ist noch ein Element auf dem Stack oder nicht.
- private Drehung invertiereDrehung(Drehung drehung) hinzugefügt; die Methode dreht die Richtung der übergebenen Drehung um,damit die Ausgangsdrehung durch die neue Drehung rückgängig gemacht werden kann, wobei drehung die Drehung die umgekehrt werden soll,ist.

2.2.4 MusikWiedergabe

- private static final String MEDIA_TYPE hinzugefügt; die Konstante speichert den Typ der Wiedergabe Datei.
- private VolumeControl volumeControl hinzugefügt; das Attribut speichert die Zugriffsinstanz für die Lautstärke-Steuerung.
- private Player player hinzugefügt; Attribut mit dem Interface player.

2.2.5 SpeicherManager

Assoziation zu XMLManager weggefallen, da diese Klasse nicht implementiert wurde.
neue Attribute:

- private static SpeicherManager instanz - Speichert die einzige Instanz dieser Klasse (Singleton).

neue Konstanten:

- private static final String EINSTELLUNGEN_RECORDSTORE
- private static final String SPIEL_RECORDSTORE
- private static final String BESTENLISTE_RECORDSTORE
- private static final String MUSIKDATEI_STANDARD
- private static final String MUSIKDATEI_HIGHSCORE

neue Methoden:

- public static SpeicherManager gibInstanz() - Singleton-"Konstruktor"
- private SpeicherManager()
- public ListenEintrag[] getBestenliste() - Rückgabetyp von BestenlisteManager auf Array von Listeneinträgen geändert.
- public void speichereBestenliste(ListenEintrag[] bestenliste) - Parameter von BestenlisteManager auf Array von Listeneinträgen geändert.
- public Einstellungen getEinstellungen() - Rückgabetyp von EinstellungenManager auf Einstellungen geändert.
- public void speichereEinstellungen(Einstellungen einstellungen) - Parameter von EinstellungenManager auf Einstellungen geändert.
- private void loescheRecordStore(String rs) - Hilfsmethode zum Löschen eines Recordstores
- public void loescheSpiel() - Löscht den Spielspeicher.
- public void loescheEinstellungen() - Löscht die Einstellungen.
- public void loescheBestenliste() - Löscht die Bestenliste.
- private DataInputStream leseRecordStore(String rs) - Hilsmethode zum Öffnen eines Recordstores.
- private void schreibeDaten(byte[] data, String rs) - Hilfsmethode zum Schreiben eines Byte-Arrays.

2.2.6 SpielController

- Die Klasse GUISpielmenue wurde komplett im SpielController integriert. Hierzu kamen alle Methoden und Attribute der GUISpielmenue.
- private int aktuellerSchwierigkeitsgrad hinzugefügt; das Attribut speichert den aktuell eingestellten Schwierigkeitsgrad.
- private boolean anspruchAufHighscorePlatz hinzugefügt; das Attribut speichert, ob der Spieler einen Anspruch auf einen Platz in der Bestenliste hat oder nicht
- private boolean normalesSpiel hinzugefügt; im Attribut wird festgehalten, ob der aktuelle Schwierigkeitsgrad dem eines normalen Spieles entspricht.
- private boolean drehungAktiv hinzugefügt; das Attribut speichert, ob gerade eine Drehung durchgeführt wird, oder nicht.
- public SpielController(GUIAnzeige elternmenue,boolean bildschirmschonerModus, boolean ladeSpiel) der Konstruktor wurde geändert, da der SpielController nun auch die GUISpielmenue-Klasse integriert.
- public void starteSpiel() Methode geändert; boolean ladeSpiel gehört nun dem Konstruktor
- private void initSpiel(boolean ladeSpiel) hinzugefügt; die Hilfsmethode bereitet in Abhängigkeit vom Parameter den Start entweder des gespeicherten oder eines neuen Spieles vor.
- private void initBildschirmschoner() hinzugefügt; die Methode startet den BildschrimschonerModus.
- private void bildschirmschonerSchritt() hinzugefügt; die Methode führt einen Durchgang des Bildschirmschoners aus. Also neuen Würfel mittels Startgenerator erzeugen und anschliessend lösen.

- private void automatischesLoesen() hinzugefügt; die Methode berechnet vom momentanen Zustand ausgehend den Loesungsweg und führt diese Drehungen dann nacheinander aus. Sie wird im Bildschirmschonermodus und im Spielmodus benutzt.
- private void redo() hinzugefügt; die Methode wiederholt, falls möglich einen zuvor rückgängig gemachten Zug.
- private void undo() hinzugefügt; die Methode macht, falls möglich den vorherigen Zug rückgängig.
- private void spielSpeichern() hinzugefügt; die Methode benutzt den Speichermanager, um die relevanten Daten des aktuellen Spiels zu speichern.
- public void setTastatureingabe(int tastencode) hinzugefügt; die Methode nimmt die Tasteneigaben der M3GDarstellun entgegen und reagiert entsprechend der Eingabe und des Zustandes.
- public void setzeDrehungBeendet() hinzugefügt; die Methode wird ausgeführt, wenn der aktuelle Vector an schrittweisen Drehungen abgearbeitet ist.
- private void dreheWuerfel(Drehung drehung, boolean history) hinzugefügt; die Methode kapselt eine vom Benutzer durchgeführte Drehung. Die Drehung wird dem HistorieManager uebergeben, wenn der entsprechende Parameter true ist.
- private void spieleMusik(boolean highscore) hinzugefügt; die Methode erzeugt einen neuen SpeicherManager und MusikManager und veranlasst, dass die passende Melodie abgespielt wird.

2.2.7 SprachManager

Für jeden String am Handybildschirm wurde eine Konstante (public static final int STRING) hinzugefügt.

2.2.8 TastaturManager

- private int ebene hinzugefügt; das Attribut speichert die Ebene der gesuchten Drehung zwischen.
- public boolean istEbene(int key) hinzugefügt; die Methode gibt zurück, ob der übergebene Tastencode eine Ebene beschreibt, oder nicht. key ist der gesuchte Tastencode; die Methode liefert true, falls der Tastencode eine Ebene auswählt, false falls nicht.
- public boolean istRichtung(int key) hinzugefügt; die Methode gibt zurück, ob der übergebene Tastencode eine Richtung beschreibt, oder nicht. key ist der gesuchte Tastencode; die Methode liefert true, falls der Tastencode eine Richtung auswählt, false falls nicht.
- public boolean istEbeneGesetzt() hinzugefügt; die Methode gibt zurückt,ob bereits eine Ebene ausgewählt wurde oder nicht; die Methode liefert true falls bereits Ebene ausgewählt ist, sonst false.
- public void setzeEbene(int key) hinzugefügt; die Methode speichert die ausgewählte Ebene für das spätere Erstellen einer Drehung zwischen.key ist der Tastencode der Drehebene.
- public int gibEbene(int key) hinzugefügt; die Methode gibt zu den jeweiligen Tastatureingabe die passende Dreh-Ebene zurück. key ist der Tastencode der Drehebene.
- public Drehung erzeugeDrehung(int key) hinzugefügt; die Methode erzeugt einen neuen Vector mit der Drehung, die ausgeführt werden soll. key ist der Tastatureingabe mit Drehrichtung. Die Methode liefert einen Vector mit Drehung.
- private boolean gibRichtung(int key) hinzugefügt; die Methode gibt die Drehrichtung zurück die der übergebenen Tastatureingabe entspricht.key ist der Tastatureingabe; die Methode liefert true falls KEY_STAR, false falls KEY_POUND.

2.2.9 ZeitManager

public String toString() wurde in public void setzeZeit(char[] ausgabeString) umgewandelt, damit beim Neuzeichnen möglichst wenig neue Strings angelegt werden.

2.3 2D-View

2.3.1 BildLader

neue Fields:

- public static final int LOGO
- public static final int SCHWIERIGKEITSGRAD
- public static final int WUERFELTYP
- public static final int TON
- public static final int SPRACHE
- public static final int FARBWUERFEL
- public static final int TEXTURWUERFEL
- public static final int DEUTSCH
- public static final int ENGLISCH
- public static final int RUMAENISCH
- public static final int FRANZOESISCH

neue Attribute:

- private static final String PFAD
- private static final String[] BILDER

2.3.2 GUIAnzeige

- Sichtbarkeit aller Attribute von private auf protected geändert, so dass die Vererbung funktioniert.
- public GUIAnzeige getElternMenue() ist weggefallen, weil wegen o.s. Änderung nicht mehr notwendig.

2.3.3 GUIBestenliste

Neue Hilfsmethode: private String erstelleBestenliste() - Erstellt den Bestenlisten-String

2.3.4 GUIEinstellungen

neue Methode: public Einstellungen getEinstellungen() - Um die Einstellungen dem Starmenü zugänglich zu machen.

2.3.5 GUILademenue

neue Hilfsmethoden:

- private boolean spielGespeichert() - Prüft via SpeicherManager, ob ein Spiel gespeichert ist.
- private void aktualisiereDarstellung() - Wird von anzeigen() aufgerufen, um die Darstellung ggf. zu aktualisieren. Dies ist für den Fall notwendig, wenn am Anfang kein Spiel gespeichert war, nach der Rückkehr ins Startmenü aber dann doch.

2.3.6 GUIManager

Keine Änderungen.

2.3.7 GUIPausebildschirm

Keine Änderungen.

2.3.8 GUISchwierigkeitsgrad

Keine Änderungen.

2.3.9 GUISpielanleitung

Keine Änderungen.

2.3.10 GUISpielmenue

Ist weggefallen, da das Spielmenü komplett im Controller realisiert wurde.

2.3.11 GUISpracheinstellungen

Keine Änderungen.

2.3.12 GUIStartbildschirm

- private static final String VERSION - String statt int, da so detailliertere Infos ausgegeben werden können.
- private static final long WARTEZEIT - Zur Konfigaration der Anzeigezeit des Startbildschirms
- public void run() - Notwendig, da diese Klasse als Thread implementiert wurde.
- public GUIStartbildschirm(Display display) - Der Konstruktor ist auch neu, da ganz zu Beginn der Startbildschirm nur als einfacher Alert gedacht war.

2.3.13 GUIStartmenue

- Keine Assoziation mehr zu GUISpielmenue, da dieses weggefallen ist.
- Neue Assoziation zur Klasse SpielController.
- Neue Hilfsmethode: private List erzeugeMenueListe() - Erzeugt die Menüliste samt Piktogrammen.

2.3.14 GUITastenbelegung

Keine Assoziation mehr zu GUISpielmenue, sondern zum SpielController.

2.3.15 GUITextausgabe

Sichtbarkeit aller Attribute zwecks Vererbung von private auf protected geändert.

2.3.16 GUIToneinstellungen

Keine Änderungen.

2.3.17 GUIWuerfeltyp

Keine Änderungen.

2.4 3D-View

2.4.1 DarstellungsInterface

Keine Änderungen.

2.4.2 FarbenLader

Keine Änderungen.

2.4.3 FarbenWuerfel

Keine Änderungen.

2.4.4 KameraManager

Durch Probleme, die sich bei der Ausrichtung der Kamera in der Nähe der z-Achse ergeben haben, werden folgende zusätzlichen Methoden benötigt.

- public boolean geheZurStandardPosition() Um die Kamera zur Ausgangsposition zu bewegen.
- public void bildschirmschonerSchritt() Um die Kamera im Bildschirmschonermodus zu bewegen.
- private void setzeOrientierung() Um die problematische Ausrichtung der Kamera in der m3g-Welt zu umgehen.
- private void vektorSubtraktion(float[] vektor1, float[] vektor2, float[] vektorZurueck) Hilfsmethode für setzeOrientierung().
- private float vektorNormalisieren(float[] vektor) Hilfsmethode für setzeOrientierung().
- private void vektorKreuzProdukt(float[] vektor1, float[] vektor2, float[] vektorZurueck) Hilfsmethode für setzeOrientierung().

2.4.5 M3GDarstellung

Zur Verschönerung der M3GDarstellung (z.B. Kamerabewegung im Bildschirmschonermodus) wurden einige zusätzliche Attribute und Methoden benötigt.

- private boolean drehungAktiv Attribut. Zeigt an, ob eine Drehung aktiv ist.
- private char[] ausgabeString Attribut. Wurde zur Speicheroptimierung angelegt, damit der GC nicht so viele Objekte löschen muss.
- private void taktschrittAusfuehren()

- public void kameraRuecksetzen()
- public boolean drehungenAbgeschlossen()
- public void setzeBildschirmschnermodus()

2.4.6 M3GRubikWuerfel

Keine Änderungen.

2.4.7 NeuzeichnerThread

Keine Änderungen.

2.4.8 Richtungspfeile

Keine Änderungen.

2.4.9 RotationsManager

Durch das Problem, dass die Drehungen im endgültigen MiDlet nicht angezeigt wurden, sind folgende Attribute und Methoden dazugekommen.

- private Drehung aktuelleDrehung()
- private Vector aktuelleDrehungen()
- private boolean aktiveDrehungen()
- private int drehungsSchritte()
- public boolean drehungsSchritt()
- public boolean gibAkiveDrehungen()

2.4.10 TexturenLader

Keine Änderungen.

2.4.11 TexturenWuerfel

Keine Änderungen.

2.4.12 Umgebung

Keine Änderungen.

2.4.13 WuerfelInterface

Keine Änderungen.

2.4.14 Wuerfel

Keine Änderungen.

3 Statistik der Testfälle

In diesem Kapitel werden die Ergebnisse der JM-Unit Testklassen zusammengefasst.
Wir haben zum Testen unserer Klassen ganz gezielt nicht die auf der "Einstiegsseite zum SEP" genannte JUnit-Variante J2ME-Unit (http://j2meunit.sourceforge.net/) verwendet, da diese seit Oktober 2003 nicht mehr weiterentwickelt wird. Stattdessen haben wir die Implementierung JMUnit (http://sourceforge.net/projects/jmunit/) verwendet, die im Augenblick in Version 1.0.1 (vom Juli 2006) vorliegt und somit viel aktueller ist.

Die einzelnen Testklassen wurden in Testsuiten, die jeweils ein Package abdecken, zusammengefasst.

3.1 Model

Es wurden folgende JMUnit Tests für das Model erstellt:

- IntuitiverLoeserTest:
  - Bei der ersten Methode wurden ca. 1 000 000 zufaellige RubikWuerfel erstellt. Auf diesen Würfeln wurde der Hilfeschritt so lange angewandt bis der Würfel gelöst wurde. Dabei muss nach dem ersten Aufruf der Hilfefunktion die erste Reihe des RubikWuerfel-Objekts gelöst sein. Nach dem dritten Aufruf ist die erste Ebene komplett gelöst und nach dem vierten Aufruf sind die ersten beiden Ebenen gelöst. Diese Eigenschaften werden mit Hilfe von Methoden aus der Klasse RubikWuerfel überprüft.
    Alle diese Tests sind erfolgreich beendet worden.
  - Analog zu der ersten Methode werden auch bei der zweiten Methode ca. 1 000 000 zufällige RubikWuerfel erstellt. Auf diesen Würfel wird der Lösungsweg berechnet und angewandt. Ist danach der RubikWuerfel gelöst, so ist der Test erfolgreich beendet. Auch hier liegt die Erfolgsquote bei 100%.
- IntuitiverLoeserEbenenTest testet die public-Methode nachsterZug der Klasse IntuitiverLoeser, wobei bei Aufruf der Hilfefunktion bereits immer eine Ebene geloest ist und diese Ebene nach oben (d.h. zur Ebene 1 wird) gesetzt wird. Hier gibt es sechs verschiedene Methoden, für jede Ebene eine. Jede Methode ist gleich aufgebaut: In der MethodeX (1 <= X <=6) ist immer die EbeneX gelöst. Dann gibt es zwei Fälle:
  - 1.Fall: Es gibt noch eine weitere EbeneY (X <Y), die gelöst ist
  - 2.Fall: Die EbeneX ist die einzige gelöste Ebene im RubikWuerfel-Objekt.

  Um diese Tests durchzuführen wurden bestimmte Drehungen auf den RubikWuerfel angewandt. Der IntuitiverLoeserEbenenTest wurde zu 100% erfüllt.
- Die Klasse IntuitiverLoeserReihenTest testet die public-Methode nachsterZug der Klasse IntuitiverLoeser, wobei bei Aufruf der Hilfefunktion bereits eine Reihe geloest ist und diese Reihe nach oben (d.h. zur Reihe 1 wird) gesetzt wird. Nach Aufruf der Hilfefunktion sind dann mindestens die Teilwuerfel in der Ebene 1 bis Stelle 6 richtig. Falls bereits bei Aufruf der Methode naechsterZug eine Ebene existiert, in der die ersten sechs Teilwuerfel richtig liegen, dann ist nach Aufruf der Methode naechsterZug bereits die erste Ebene richtig geloest. Diese Eigenschaft wird immer mitgetestet. In dieser Testklasse gibt es 12 Methoden, die wie bei der Klasse IntuitiverLoeserEbenenTest gleich aufgebaut sind. In der MethodeX ist immer die ReiheX gelöst. Es werden in jeder Methode vier Fälle unterschieden (Jede ReiheX liegt in zwei Ebenen. Hier werden diese Ebenen mit EbeneY und EbeneZ bezeichnet) :
  - 1.Fall: Die EbeneY ist im Loesungsprozess weiter fortgeschritten als die EbeneZ.
  - 2.Fall: Die Ebene Z ist im Loesungsprozess weiter fortgeschritten als die Ebene Y.
  - 3.Fall: Es ist noch die Reihe A geloest, aber die dazugehoerigen Ebenen sind noch nicht so weit mit den Loesungsprozess fortgeschritten als eine Ebene, die zur ReiheX gehoert. (A<X)

- 4.Fall: Nur die ReiheX ist geloest.

Um diesen Test durchzuführen wurden bestimmte Drehungen auf den RubikWuerfel angewandt. Der IntuitiverLoeserReihenTest wurde zu 100% erfüllt.

- Die Klasse RubikWuerfelTest testet alle public-Methoden der Klasse RubikWuerfel. Wobei die Methoden testeSetzeGeloestEbeneOben() und testeSetzeGeloestReiheOben() noch vertieft in den Klassen IntuitiverLoeserEbenenTest und IntuitiverLoeserReihenTest geprüft werden.
  Der RubikWurfelTest wurde erfolgreich beendet.

Alle Testfälle wurden erfolgreich beendet. Siehe dazu Abbildung 2.

Listing 1: IntuitiverLoeserTest.java

```
1  package testcases;
2
3  import jmunit.framework.cldc11.AssertionFailedException;
4  import jmunit.framework.cldc11.TestCase;
5  import yarc.model.*;
6  import java.util.*;
7
8  /**
9   * Die Klasse <code>IntuitiverLoeserTest</code> testet die public-Methoden
10  * <code>berechneLoesungsweg</code> und <code>nachsterZug</code> der Klasse
11  * <code>IntuitiverLoeser</code>
12  *
13  * @author Christina Bloechl
14  *
15  */
16 public class IntuitiverLoeserTest extends TestCase {
17
18         public IntuitiverLoeserTest() {
19                 super(2, "IntuitiverLoeserTest");
20         }
21
22         public void test(int testNr) throws Throwable {
23                 switch (testNr) {
24                 case 0:
25                         testeNaechsterZug();
26                         break;
27                 case 1:
28                         testeBerechneLoesungsweg();
29                         break;
30                 }
31         }
32
33         /**
34          * Die Methode <code>testeNaechsterZug</code> testet die Methode
35          * <code>naechsterZug</code> der Klasse <code>IntuitiverLoeser</code>.
36          * Dabei wird in einer for-Schleife 1000 mal mit Hilfe des Startgenerators
37          * ein zufaellig verdrehter Wuerfel erzeugt und auf diesen dann so oft die
38          * Methode <code>nachsterZug</code> angewendet, bis der Wuerfel geloest ist.
39          *
40          * @throws AssertionFailedException
41          */
42         public void testeNaechsterZug() throws AssertionFailedException {
43                 RubikWuerfel wuerfel = new RubikWuerfel();
44                 IntuitiverLoeser loeser = new IntuitiverLoeser();
```

```
45              Startgenerator start = new Startgenerator(5);
46
47              for (int i = 0; i < 1000; i++) {
48                      Vector drehungen = start.erzeugeWuerfel();
49                      wuerfel.dreheEbene(drehungen);
50                      /*
51                       * Nach einem Loesungsschritt muss eine Reihe geloest sein.
52                       */
53                      loeser.naechsterZug(wuerfel);
54                      assertTrue("Fehler beim Test: " + i + " in testeNaechsterZug."
55                                                      ,wuerfel.eineReiheGeloest());
56
57                      /*
58                       * Nach weiteren zwei Loesungsschritten muss die erste Ebene
59                       * geloest sein.
60                       */
61                      loeser.naechsterZug(wuerfel);
62                      loeser.naechsterZug(wuerfel);
63                      assertTrue("Fehler beim Test: " + i + " in testeNaechsterZug."
64                                                      ,wuerfel.istEineEbeneGeloest());
65
66                      /*
67                       * Nach einem weiteren Loesungsschritt muss die zweite Ebene
68                       * geloest sein.
69                       */
70                      loeser.naechsterZug(wuerfel);
71                      assertTrue("Fehler beim Test: " + i + " in testeNaechsterZug."
72                                              ,wuerfel.sindZweiEbenenGeloest());
73
74                      while (!wuerfel.istGeloest()) {
75                              loeser.naechsterZug(wuerfel);
76                      }
77                      assertTrue("Fehler beim Test: " + i + " in testeNaechsterZug."
78                                                              , wuerfel.istGeloest());
79              }
80      }
81
82      /**
83       * Die Methode <code>testeBerechneLoesungsweg</code> testet die Methode
84       * <code>berechneLoesungsweg</code> der Klasse <code>IntuitiverLoeser</code>.
85       * Dabei wird in einer for-Schleife 1000 mal mit Hilfe des Startgenerators
86       * ein zufoellig verdrehter Wuerfel erzeugt und auf diesen dann die Methode
87       * <code>berechneLoesungsweg</code> angewendet, so dass der Wuerfel geloest ist.
88       * @throws AssertionFailedException
89       */
90      public void testeBerechneLoesungsweg() throws AssertionFailedException {
91              RubikWuerfel wuerfel = new RubikWuerfel();
92              IntuitiverLoeser loeser = new IntuitiverLoeser();
93              Startgenerator start = new Startgenerator(5);
94
95              for (int i = 0; i < 1000; i++) {
96                      Vector drehungen = start.erzeugeWuerfel();
97                      wuerfel.dreheEbene(drehungen);
98                      loeser.berechneLoesungsweg(wuerfel);
99                      assertTrue("Fehler beim Test: " + i
100                             + " in testeBerechneLoesungsweg", wuerfel.istGeloest());
101             }
```

```
102             }
103
104 }
```

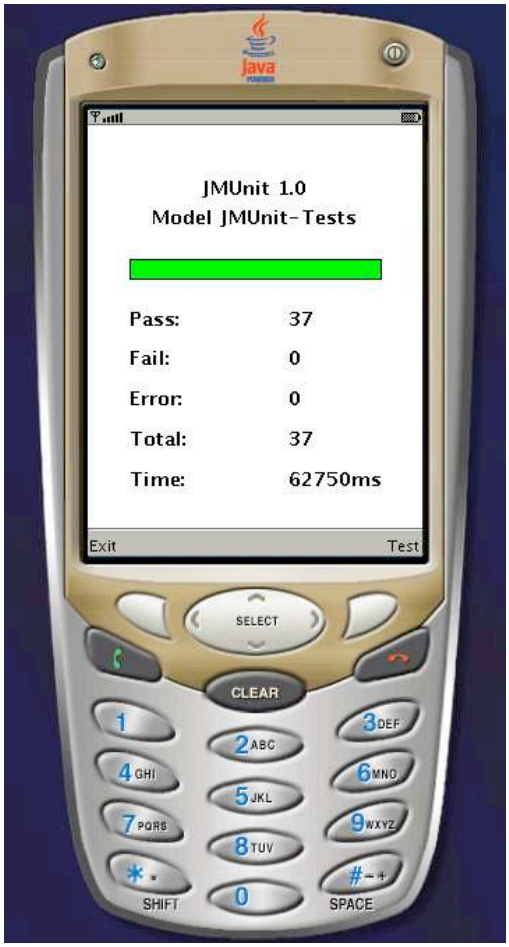


Abbildung 2 - TestSuiteModel

3.2 Controller

Im Controller wurden die Folgenden Testfälle in ein TestSzenario zusammengefasst:

- EinstellungenManagerTest
- HistorieManagerTest
- SpeicherManagerTest
- ZeitManagerTest

Alle Testfälle wurden erfolgreich beendet. Siehe dazu Abbildung 3.

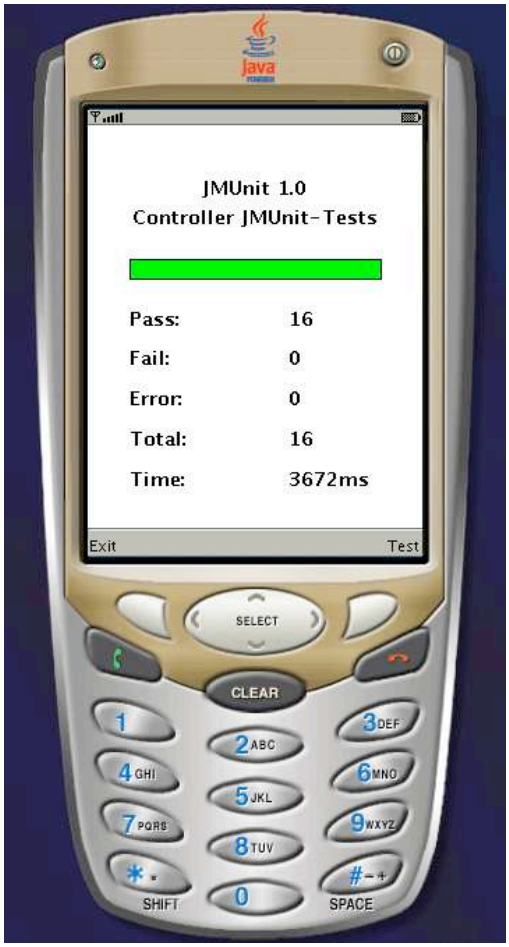


Abbildung 3 - TestSuiteController

3.3 View

JM-Unit-Testfälle für Klassen der View waren nahezu unmöglich, da die View-Klassen eine Zugriff auf die Canvas benötigen, diese aber durch die Fortschrittsanzeige von JM-Unit belegt ist.
Daher wurden durch HilfstestKlassen (nicht JM-Unit) systematische Sichttests gemacht, um eine fehlerfreie Funktion der gestesteten View-Klassen zu gewährleisten. Vereinzelt (wo möglich z.B. BildLader) wurden auch JM-Unit TestKlassen angelegt.
Alle Klassen haben die Tests bestanden.

Abbildung 4 - Startmenü

Abbildung 5 - Ton Einstellungen

Abbildung 6 - Farbenwürfel

Abbildung 7 - Texturenwürfel

3.4 Zusammenfassung

Abschließend ist zu bemerken, dass alle Pflich- und Wunschkriterien des Pflichtenheftes vollständig implementiert und auf ihre Funktionalität getestet wurden. Der aktuelle Versionsstand: Revision 273[1] ist sowohl in der Emulationsumgebung KToolbar als auch auf passenden Handys lauffähig (wurde getestet).

[1] Die Revisionsnummer wäre eigentlich deutlich höher, aber durch das Update des SVN-Servers mit daraus resultierenden Neu-Anlegen des Repositums wurde die Revisionsnummer zurückgesetzt.

4 Implementierungsplan

In diesem Kapitel wird beschrieben, wie die Implementierung der einzelnen Klassen zeitlich verlaufen ist. Dabei wird besonders darauf eingegangen, welche Unterschiede sich im Vergleich zur Planung in der Feinspezifikationsphase ergeben haben.

4.1 Zeitplan - Feinspezifikation