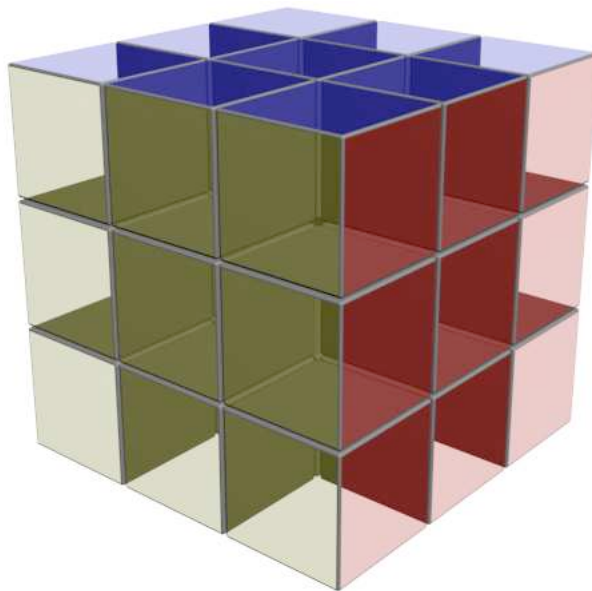




»Validierung« v 1.0

*YaRC*

Yet another Rubik Cube



Phase	Phasenverantwortliche(r)	E-Mail

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Testszenarien</b>	<b>4</b>
2.1	Testszenario 1 - Ein normales Spiel . . . . .	4
2.2	Testszenario 2 - Vollautomatisches Lösen . . . . .	4
2.3	Testszenario 3 - Dauerhilfe . . . . .	4
2.4	Testszenario 4 - Alles rückgängig . . . . .	5
2.5	Testszenario 5 - Bildschirmschoner . . . . .	5
2.6	Testszenario 6 - Highscore geknackt* . . . . .	5
2.7	Testszenario 7 - Speichern und Laden* . . . . .	6
2.8	Testszenario 8 - Wunschfarben und -lautstärke* . . . . .	6
2.9	Testszenario 9 - Anruf* . . . . .	7
2.10	Testszenario 10 - Mehrsprachig* . . . . .	7
2.11	Zusammenfassung . . . . .	7
<b>3</b>	<b>Coverage-Tests</b>	<b>8</b>
3.1	Abdeckung der JM-Unit Testfälle . . . . .	8
3.1.1	Model . . . . .	9
3.1.2	Controller . . . . .	9
3.2	Komplette Abdeckung alle Testfälle . . . . .	9
3.2.1	Model . . . . .	9
3.2.2	Controller . . . . .	9
3.2.3	Zweidview . . . . .	9
3.2.4	Dreidview . . . . .	10
<b>4</b>	<b>Regressionstests</b>	<b>11</b>
<b>5</b>	<b>Beschreibung Fehler</b>	<b>14</b>
<b>6</b>	<b>Anhang</b>	<b>15</b>
6.1	Cobertura Reports für JM-Unit . . . . .	15
6.1.1	Model . . . . .	15
6.1.2	Controller . . . . .	16
6.2	Cobertura Reports für alle Testfälle . . . . .	16
6.2.1	Model . . . . .	17
6.2.2	Controller . . . . .	18
6.2.3	2D View . . . . .	19
6.2.4	3D View . . . . .	20
	<b>Glossar</b>	<b>21</b>

---

# 1 Einleitung

In diesem Dokument wird die Validierungsphase des SEP-Projekts *Rubikwürfel auf J2ME-Handy* beschrieben. Durch umfangreiche Testabläufe wird gewährleistet, dass der in der Implementierungsphase erarbeitete Code, korrekt und stabil ist. Dabei wird in folgenden Schritten vorgegangen.

- Die zuvor im Pflichtenheft ausgearbeiteten Tests und Testszenarien werden abgearbeitet, um festzustellen, ob der Code korrekt ist.
- Um zu überprüfen, wieviel des erarbeiteten Codes durch die Testszenarien abgedeckt werden, wird ein [Line Coverage](#) und [Branch Coverage](#) Test ausgeführt.
- Damit nach möglichen Änderungen am Quellcode wieder sofort überprüft werden kann, ob der Algorithmus, der den Rubikwürfel löst korrekt läuft, werden Regressionstest geschrieben. Durch deren Ausführen kann der Entwickler feststellen, ob der Algorithmus noch korrekt läuft.

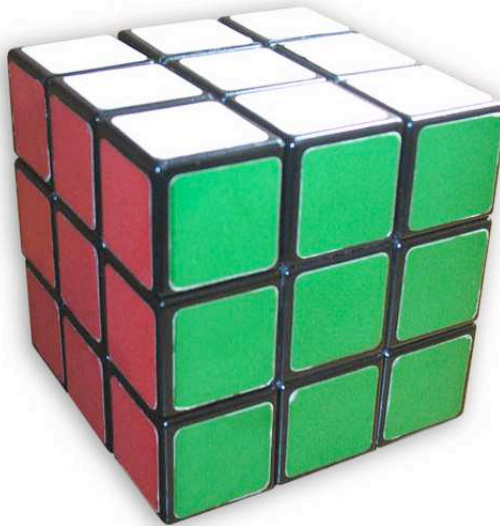


Abbildung 1 - Rubik-Würfel

## 2 Testszenarien

Die Testszenarien setzen sich aus den im Pflichtenheft genannten Testfällen (TF) zusammen. Mit Stern (\*) markierte Szenarien enthalten erweiterte Testfälle bzw. Funktionen.

### 2.1 Testszenario 1 - Ein normales Spiel

Ein unerfahrener Benutzer startet das Spiel und möchte (unter Zuhilfenahme aller verfügbaren Spielhilfen) einen Würfel lösen, was er schließlich auch schafft.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T020/	Aufrufen der Spielanleitung.	OK
/T030/	Aufrufen der Tastenbelegung aus dem Startmenü.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T100/	Aufrufen der Hilfefunktion.	OK
/T090/	Aufrufen der Tastenbelegung aus dem Spielmenü.	OK
/T120/	Rückgängig machen eines ausgeführten Zuges.	OK
/T130/	Zuvor rückgängig gemachten Zug wiederherstellen.	OK
	Wiederholen von 5. bis 9. bis zur Lösung des Würfels.	OK
/T160/	Beenden des Programmes.	OK

### 2.2 Testszenario 2 - Vollautomatisches Lösen

Der Benutzer startet das Spiel, lässt den Würfel aber nach ein paar Versuchen automatisch lösen.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 3. bis 5.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T110/	Aufrufen des automatischen Lösens.	OK
/T160/	Beenden des Programmes.	OK

### 2.3 Testszenario 3 - Dauerhilfe

Der Benutzer startet ein neues Spiel und löst dieses ausschließlich durch wiederholtes Aufrufen der Hilfefunktion.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T100/	Aufrufen der Hilfefunktion.	OK
	Wiederholen von 3. und 4. bis zur Lösung des Würfels.	OK
/T160/	Beenden des Programmes.	OK

## 2.4 TestszENARIO 4 - Alles rückgängig

Der Benutzer beginnt ein neues Spiel und macht genau zwei Züge. Dann möchte er durch wiederholtes Aufrufen der **undo**-Funktion alle seine Züge rückgängig machen, wobei er versucht, die **undo**-Funktion öfter als zwei mal aufzurufen. Im Anschluss daran ruft er das automatische Lösen auf, welches er aber, nachdem er es gestartet hat, abubrechen versucht.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Einmaliges Wiederholen von 3. bis 5.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T120/	Rückgängig machen eines ausgeführten Zuges.	OK
	Mind. drei mal Wiederholen von 7. und 8.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T110/	Aufrufen des automatischen Lösens.	OK
	Drücken aller Tasten, die aus Benutzersicht evtl. zum (nicht möglichen) "Abbrechen" der Aktion führen könnten.	OK
/T160/	Beenden des Programmes.	OK

## 2.5 TestszENARIO 5 - Bildschirmschoner

Der Benutzer beginnt ein Spiel, welches er aber abbricht, um danach den Bildschirmschoner zu starten. Nachdem der Bildschirmschoner mind. 10 Minuten gelaufen ist, beendet er diesen und danach das Programm.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 3. bis 5.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T140/	Spiel beenden.	OK
/T150/	Aufrufen der Bildschirmschonerfunktionalität.	OK
	Mind. 10 Minuten warten	OK
/T160/	Beenden des Programmes.	OK

## 2.6 TestszENARIO 6 - Highscore geknackt\*

Der Benutzer betrachtet nach Programmstart die (nicht leere) **Highscore**-Tabelle, stellt den höchsten Schwierigkeitsgrad ein und beginnt dann ein neues Spiel. Nach ein paar Zügen pausiert er das Spiel für drei Minuten und setzt es danach fort. Er löst den Würfel in "Rekordzeit", eine Melodie ertönt und er erhält den höchsten Rang in der Highscore-Tabelle. Zufrieden beendet er das Programm.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T240/	Anzeigen der Highscore-Tabelle.	OK
/T190/	Ändern des Schwierigkeitsgrades.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 5. bis 7.	OK
/T250/	Spiel pausieren.	OK
	Drei Minuten warten.	OK
/T260/	Pause beenden.	OK
	Wiederholen von 5. bis 7.	OK
/T280/	Erreichen eines Highscore-Tabellenplatzes.	OK
/T290/	Eintragen in die Highscore-Tabelle.	OK
/T160/	Beenden des Programmes.	OK

## 2.7 TestszENARIO 7 - Speichern und Laden\*

Der Benutzer beginnt ein neues Spiel mit zuvor selbst gewählten Themenwürfel. Nach ein paar Zügen und der Benutzung der Hilfefunktion speichert er den Spielstand und beendet sowohl Spiel als auch Programm. Nach erneutem Programmstart lädt er das gespeicherte Spiel und setzt es fort.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T210/	Auswählen des <a href="#">Themenwürfels</a> .	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 4. bis 6.	OK
/T080/	Aufrufen des Spielmenüs.	OK
/T100/	Aufrufen der Hilfefunktion.	OK
/T270/	Speichern des aktuellen Spielzustandes und Beenden des Spiels.	OK
/T160/	Beenden des Programmes.	OK
/T010/	Das Programm starten.	OK
/T170/	Laden eines gespeicherten Spiels.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 14. bis 16. bis zum Lösen des Würfels.	OK
/T160/	Beenden des Programmes.	OK

## 2.8 TestszENARIO 8 - Wunschfarben und -lautstärke\*

Der Benutzer wählt vor Spielstart den Farbwürfel aus, setzt die Lautstärke auf ein Maximum und schaltet - paradoxerweise - den Ton dann aus. Beim Lösen des folgenden normalen Spiels ist dann auch keine Melodie zu hören.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T200/	Wählen des Farbwürfels.	OK
/T230/	Ändern der Lautstärke.	OK
/T180/	Ändern der Ton-Einstellungen (aus).	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 6. bis 8. bis zum Lösen des Würfels.	OK
/T160/	Beenden des Programmes.	OK

## 2.9 Testszenario 9 - Anruf\*

Der Benutzer startet ein normales Spiel. Während des Spiels wird er angerufen. Nach dem Anruf möchte der Benutzer das durch den Anruf automatisch pausierte Spiel fortsetzen.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T040/	Ein neues Spiel beginnen.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 3. bis 5.	OK
/T310/	Anruf während Ausführung des Programms.	OK
	Telefonat beenden.	OK
	Pause beenden.	OK
/T050/	Freies Drehen des Würfels.	OK
/T060/	Auswählen einer zu drehenden Ebene und der Drehrichtung.	OK
/T070/	Drehen der ausgewählten Ebene.	OK
	Wiederholen von 12. bis 14. bis zum Lösen des Würfels.	OK
/T160/	Beenden des Programmes.	OK

## 2.10 Testszenario 10 - Mehrsprachig\*

Der Benutzer möchte die Standardsprache (Deutsch) des Spieles auf Chinesisch ändern. Ist kein Chinesisch vorhanden, so wählt der Benutzer Englisch aus und spielt ein normales Spiel (Testszenario 1). Hierbei werden jetzt alle sprachlichen Elemente (Menüs, Anleitungen etc.) auf Englisch angezeigt.

TF	Bezeichnung	Bestanden
/T010/	Das Programm starten.	OK
/T220/	Sprach-Einstellungen ändern.	OK
	Weiter bei Testszenario 1 ab Schritt 2.	OK

## 2.11 Zusammenfassung

Zusammenfassend lässt sich anmerken, dass alle Testszenarien fehlerlos durchlaufen wurden.

## 3 Coverage-Tests

Für unserer Coverage-Tests verwenden wir das Tool Cobertura for J2ME v1.0.2 (02.05.2006) <http://cobertura4j2me.dreameffect.org/> von Ludovic Dewailly. Dieser hat das freie "Test Coverage Tool" Cobertura v1.2 for J2SE <http://cobertura.sourceforge.net/> für die Nutzung in J2ME -Projekten modifiziert. Abbildung 2 zeigt das Emulationsfenster der KToolbar mit dem Symbol für den "trusted" Schreibzugriff von Cobertura auf die Festplatte (/root1) des Handys.

Dabei werden folgende Bibliotheken verwendet:

- Apache Ant (Java-based build tool)
- ASM (Java bytecode manipulation framework)
- JavaNCSS (Source measurement suite for Java)
- GNU getopt (Java command line option parser)



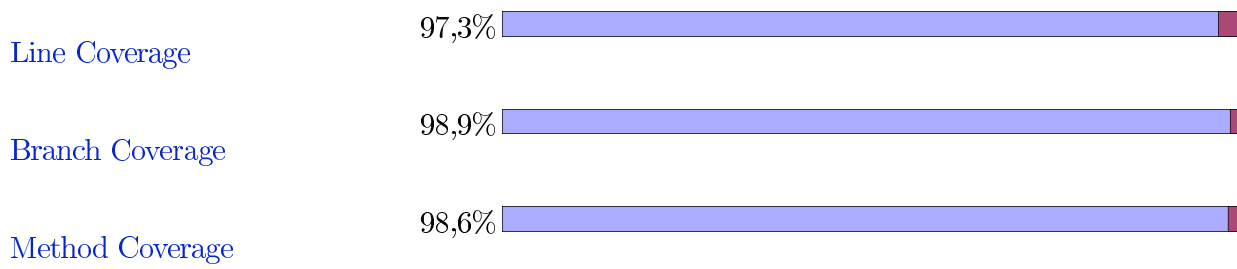
Abbildung 2 - Emulationsfenster mit Cobertura for J2ME

### 3.1 Abdeckung der JM-Unit Testfälle

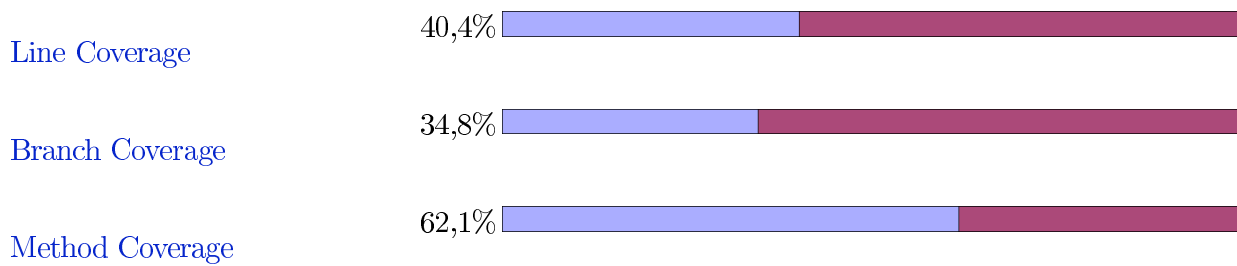
Für den Code Coverage Test der JM-Unit Testfälle (decken nur Model und Controller ab!) wurde eine Testsuite erstellt, die alle entsprechenden Testfälle aufruft. Diese wurde mit der Cobertura for J2ME-Erweiterung ergänzt und ausgeführt. Bedingt durch die Komplexität der Testklassen und dem permanenten Schreibzugriff dauerte der Durchlauf mehrere Stunden. Die Ergebnisse sind wie folgt aufgeschlüsselt:



### 3.1.1 Model

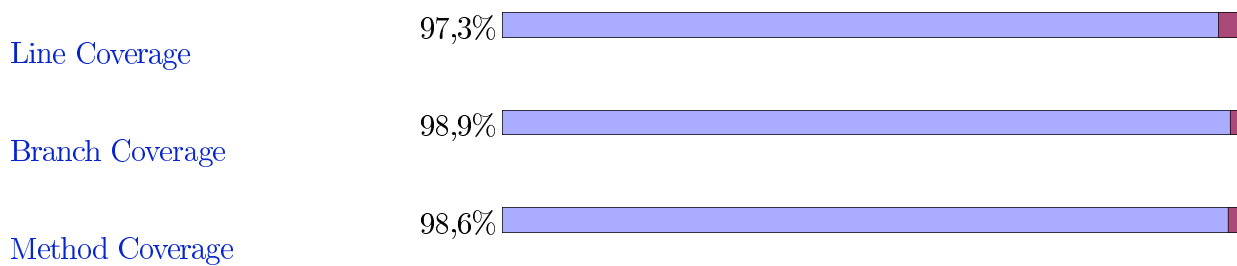


### 3.1.2 Controller

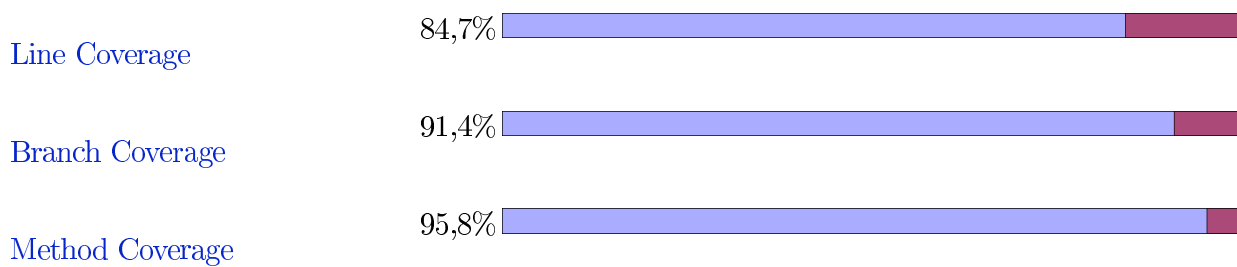


## 3.2 Komplette Abdeckung alle Testfälle

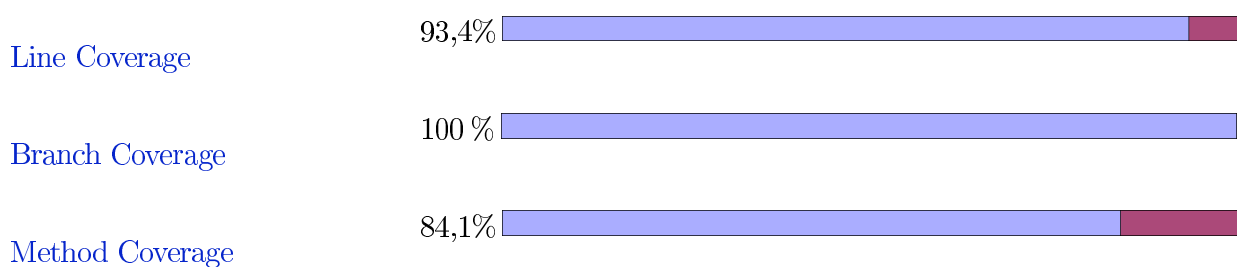
### 3.2.1 Model



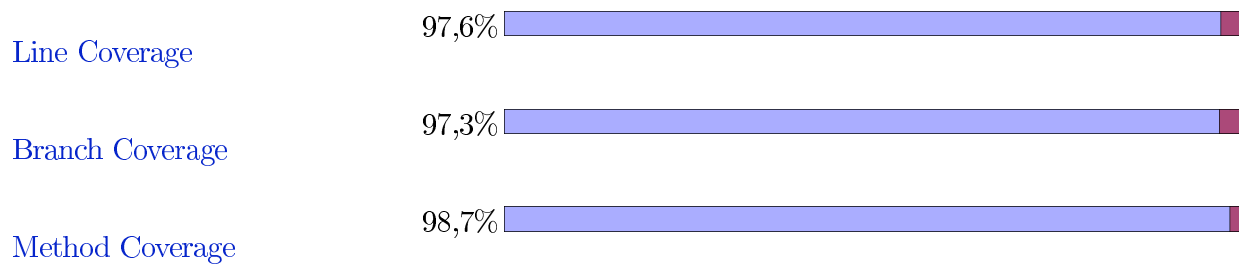
### 3.2.2 Controller



### 3.2.3 Zweidview



#### 3.2.4 Dreidview



Im Anhang (ab Seite [15](#)) befinden sich die einzelnen Reports, die Cobertura for J2ME v1.0.2 erzeugt hat.

---

## 4 Regressionstests

Es wurden folgende JUnit Tests für das Model erstellt. Diese Tests prüfen die Schnittstelle zwischen Controller und Model:

- **IntuitiverLoeserTest:**

- Bei der ersten Methode wurden ca. 100 zufällige **RubikWuerfel** erstellt. Auf diesen Würfeln wurde der Hilfeschnitt 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. 100 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 **naechsterZug** der Klasse **IntuitiverLoeser**, wobei bei Aufruf der Hilfefunktion bereits immer eine Ebene gelöst 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** ( $1 \leq X \leq 6$ ) 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 **naechsterZug** der Klasse **IntuitiverLoeser**, wobei bei Aufruf der Hilfefunktion bereits eine Reihe gelöst 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 gelöst. 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 Lösungsprozess weiter fortgeschritten als die **EbeneZ**.
- 2.Fall: Die **EbeneZ** ist im Lösungsprozess weiter fortgeschritten als die **EbeneY**.
- 3.Fall: Es ist noch die **ReiheA** gelöst, aber die dazugehörigen Ebenen sind noch nicht so weit mit den Lösungsprozess fortgeschritten als eine Ebene, die zur **ReiheX** gehört. ( $A < X$ )
- 4.Fall: Nur die **ReiheX** ist gelöst.

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

Es wurden folgende JUnit Tests für das Model erstellt:

- **IntuitiverLoeserTest:**

- Bei der ersten Methode wurden ca. 1 000 000 zufällige **RubikWuerfel** erstellt. Auf diesen Würfeln wurde der Hilfeschrift 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 **naechsterZug** der Klasse **IntuitiverLoeser**, wobei bei Aufruf der Hilfefunktion bereits immer eine Ebene gelöst 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** ( $1 \leq X \leq 6$ ) 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 **naechsterZug** der Klasse **IntuitiverLoeser**, wobei bei Aufruf der Hilfefunktion bereits eine Reihe gelöst ist und diese Reihe nach oben (d.h. zur **Reihe 1** wird) gesetzt wird. Nach Aufruf der Hilfefunktion sind dann mindestens die Teilwürfel in der Ebene 1 bis Stelle 6 richtig. Falls bereits bei Aufruf der Methode **naechsterZug** eine Ebene existiert, in der die ersten sechs Teilwürfel richtig liegen, dann ist nach Aufruf der Methode **naechsterZug** bereits die erste Ebene richtig gelöst. 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 Lösungsprozess weiter fortgeschritten als die **EbeneZ**.
  - 2.Fall: Die **EbeneZ** ist im Lösungsprozess weiter fortgeschritten als die **EbeneY**.
  - 3.Fall: Es ist noch die **ReiheA** gelöst, aber die dazugehörigen Ebenen sind noch nicht so weit mit den Lösungsprozess fortgeschritten als eine Ebene, die zur **ReiheX** gehoert. ( $A < X$ )
  - 4.Fall: Nur die **ReiheX** ist gelöst.

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 **RubikWuerfelTest** wurde erfolgreich beendet.
- Die Klasse **BestenlisteBerechnenTest** testet die **public**-Methode **berechnePunkte** der Klasse **BestenlisteBerechnen**.

---

Es wird überprüft, ob die berechneten Punkte (abhängig vom Schwierigkeitsgrad und der Zeit) mit den erwarteten Punkten übereinstimmt.

- Die Klasse **StartgeneratorTest** testet die **public**-Methode **erzeugeWuerfel** der Klasse **Startgenerator**. Es wird für jeden Schwierigkeitsgrad getestet, ob der erzeugte **Vector** von Drehungen die richtige Größe hat und es wird sichergestellt, dass durch diese Drehungen kein gelöster Würfel erzeugt wird. Außerdem wird überprüft, ob der erzeugte Würfel stets lösbar ist.

Da es sich bei den JM-Unit Tests für Regressionsaufgaben um die selben handelt, die auch schon in der Implementierungs- und Validierungsphase die Korrektheit des Models garantiert haben, sind auch die Code-Coverage Tests dieselben. Daher sei hierfür nur auf das Kapitel 3.1 (Seite [8](#)) verwiesen, in dem diese Tests behandelt wurden.

## 5 Beschreibung Fehler

- **1. Fehler:** Aufruf von Hilfeschrift oder Automatisches Lösen bei gelöstem Würfel (nur im unverdrehten Modus möglich)  
⇒ `NoSuchElementException`, aber das Programm stürzte trotzdem nicht ab. Es kam bei den nächsten Drehungen zu einem Fehler in der 3D-Darstellung.  
**Lösung:** Prüfung auf leeren Drehungsvektor.
- **TestszENARIO 9** (Seite 7) (Anruf während eines Spiels) konnte nur teilweise erfüllt werden. Nach einem Anruf kann zwar das Spiel problemlos fortgeführt werden, aber die Zeit wird nicht gestoppt und der Pausebildschirm wird nach dem Anruf nicht angezeigt.  
**Ursache:** Die JVM des Sony Ericsson Handys verhält sich bei einem Anruf so, dass die Java Anwendung im Hintergrund weiterläuft und es wird nicht `pauseApp()` aufgerufen. Unsere Vermutung: In `pauseApp` könnte theoretisch längerer Code ausgeführt werden und somit könnte der Anruf nicht entgegengenommen werden.

## 6 Anhang

### 6.1 Cobertura Reports für JM-Unit

In diesem Kapitel werden die Original-Reports, die "Cobertura for J2ME" während den JM-Unit Tests produziert hat, abgebildet. Diese Testüberdeckungen gelten also sowohl für die JM-Unit-Test der Implementierungsphase, als auch für den entworfenen Regressionstest der Validierungsphase.

#### 6.1.1 Model

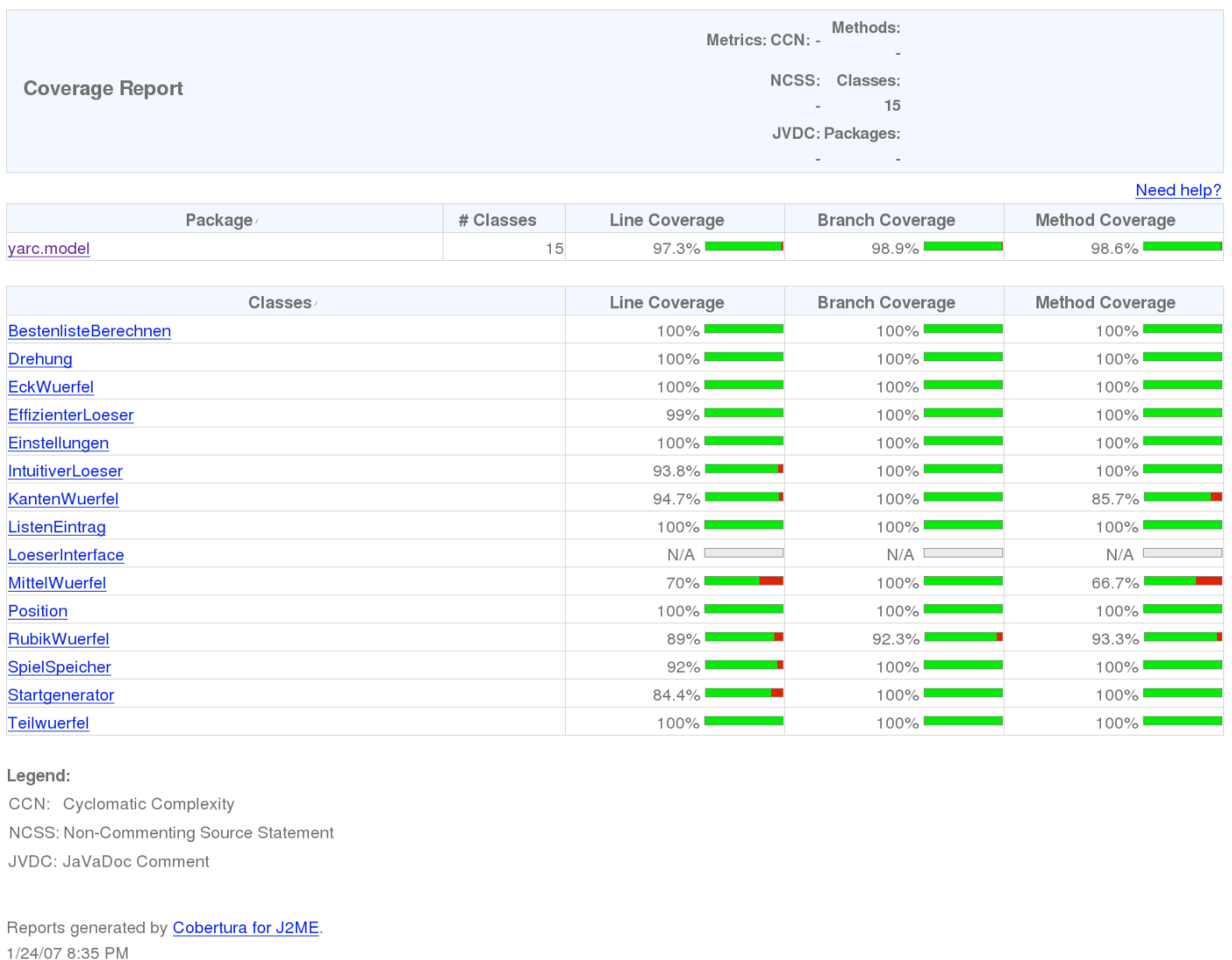


Abbildung 3 - Report der JM-Unit Testfälle (Model)

## 6.1.2 Controller

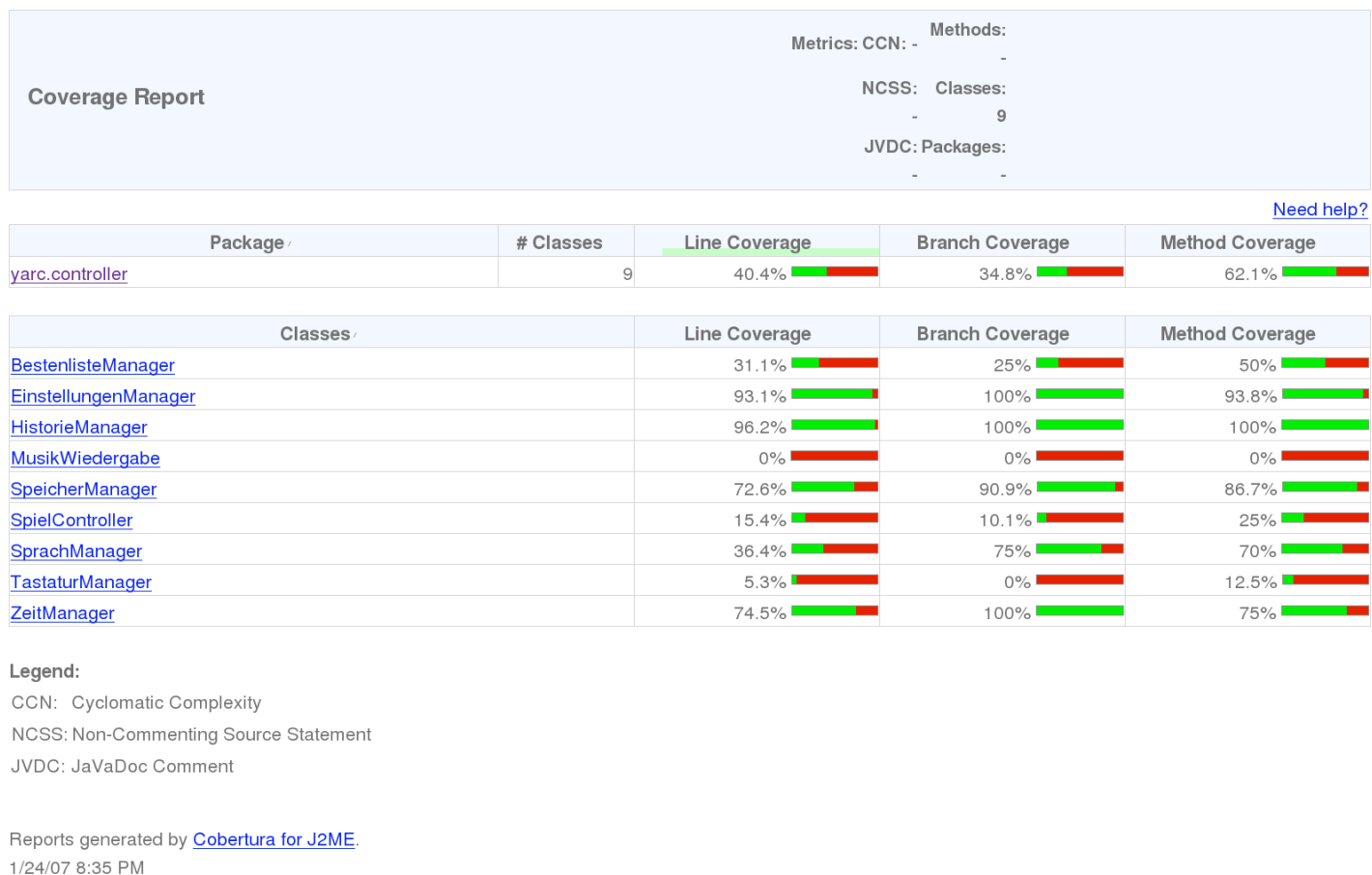


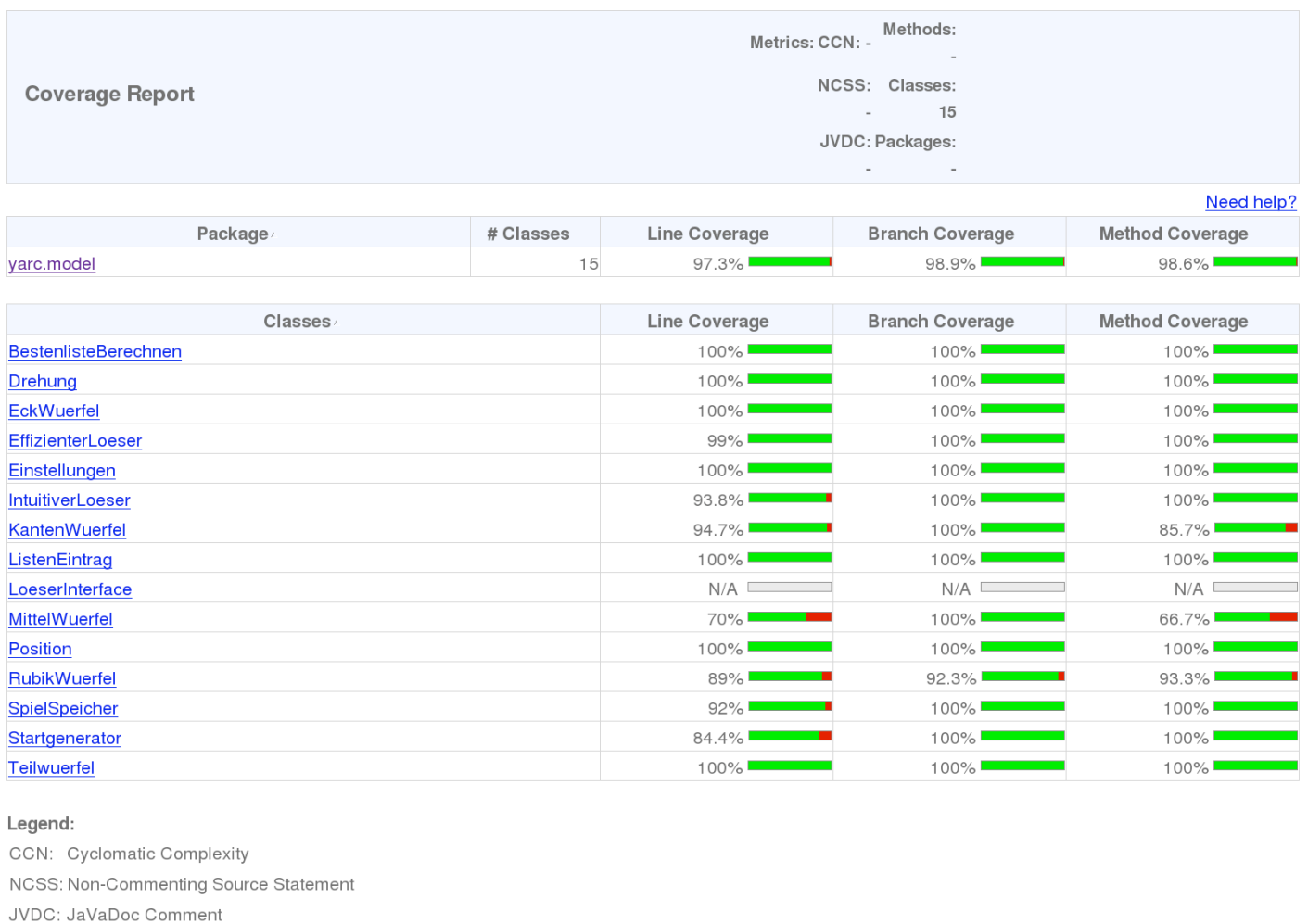
Abbildung 4 - Report der JM-Unit Testfälle (Controller)

## 6.2 Cobertura Reports für alle Testfälle

In diesem Kapitel werden die Original-Reports, die "Cobertura for J2ME" während der gesamten Testfälle (Systemintegrationstest) produziert hat, abgebildet.



## 6.2.1 Model

Reports generated by [Cobertura for J2ME](#).

1/24/07 8:35 PM

Abbildung 5 - Report aller Testfälle (Model)

## 6.2.2 Controller

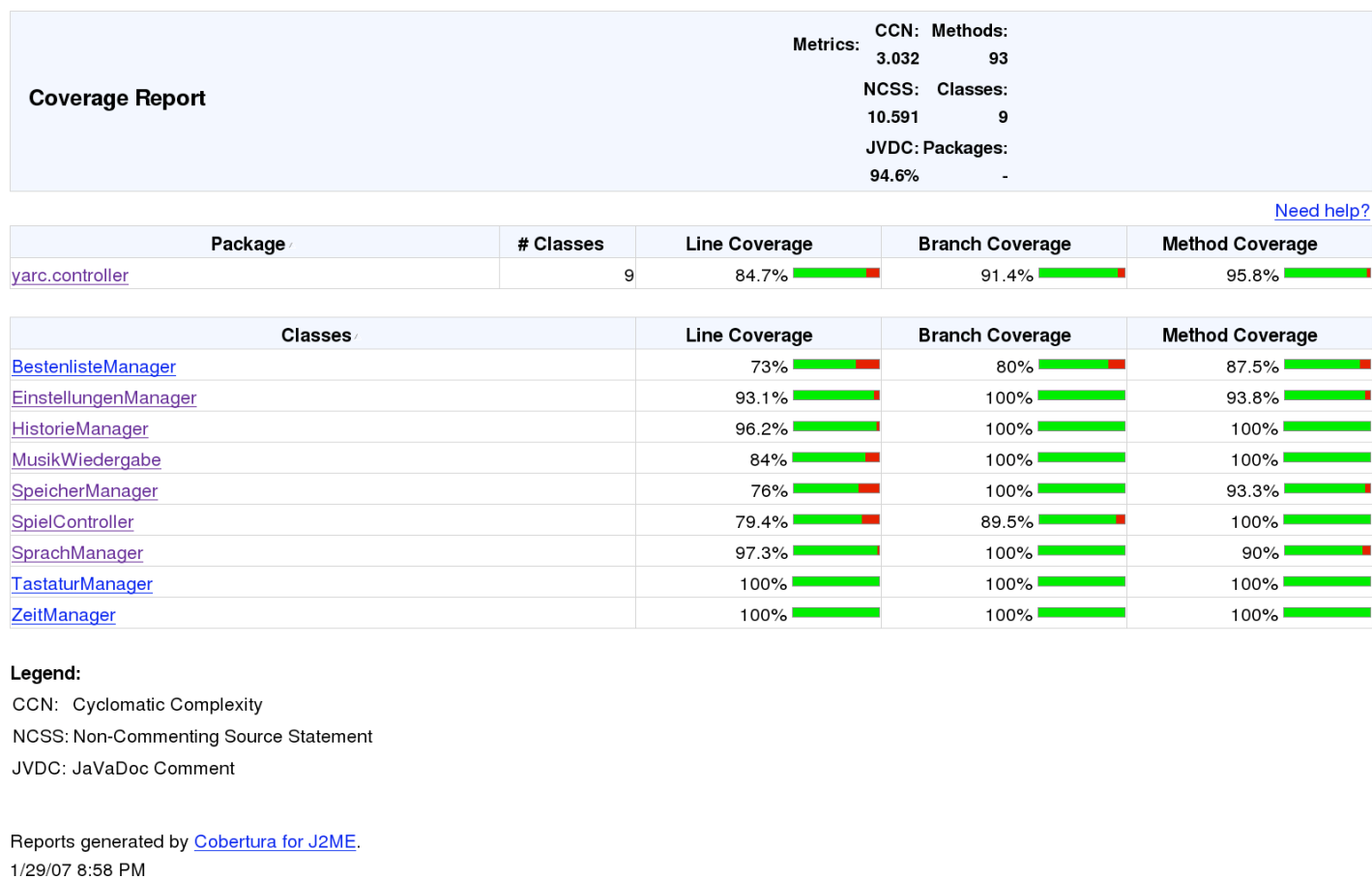


















































Abbildung 6 - Report aller Testfälle (Controller)

## 6.2.3 2D View

<b>Coverage Report</b>	<b>Metrics:</b>	<b>CCN: Methods:</b>
		1.857 63
	<b>NCSS: Classes:</b>	
	6.857	16
	<b>JVDC: Packages:</b>	
	95.2%	-

[Need help?](#)

Package	# Classes	Line Coverage	Branch Coverage	Method Coverage
<a href="#">yarc.zweidview</a>	16	93.4% 	100% 	84.1% 

Classes	Line Coverage	Branch Coverage	Method Coverage
<a href="#">BildLader</a>	90.9% 	100% 	100% 
<a href="#">GUIAnzeige</a>	100% 	100% 	100% 
<a href="#">GUIBestenliste</a>	100% 	100% 	100% 
<a href="#">GUIEinstellungen</a>	95% 	100% 	60% 
<a href="#">GUILademenue</a>	88.7% 	100% 	100% 
<a href="#">GUIManager</a>	84.2% 	100% 	71.4% 
<a href="#">GUIPausebildschirm</a>	93.3% 	100% 	75% 
<a href="#">UISchwierigkeitsgrad</a>	95.8% 	100% 	75% 
<a href="#">UISpielanleitung</a>	100% 	100% 	100% 
<a href="#">UISpracheinstellungen</a>	93% 	100% 	80% 
<a href="#">UIStartbildschirm</a>	86.7% 	100% 	100% 
<a href="#">UIStartmenue</a>	92.9% 	100% 	100% 
<a href="#">UITastenbelegung</a>	100% 	100% 	100% 
<a href="#">UITextausgabe</a>	92.3% 	100% 	75% 
<a href="#">UIToneinstellungen</a>	97.6% 	100% 	80% 
<a href="#">UIWuerfeltyp</a>	97.1% 	100% 	80% 

**Legend:**

CCN: Cyclomatic Complexity

NCSS: Non-Commenting Source Statement

JVDC: JaVaDoc Comment

Reports generated by [Cobertura for J2ME](#).

1/29/07 8:58 PM











































Abbildung 7 - Report aller Testfälle (2D View)

## 6.2.4 3D View

<b>Coverage Report</b>	Metrics:	CCN: 2.884	Methods: 69
		NCSS: 10.275	Classes: 14
		JVDC: 98.6%	Packages: -

[Need help?](#)

Package	# Classes	Line Coverage	Branch Coverage	Method Coverage
<a href="#">yarc.dreidview</a>	14	97.6% 	97.3% 	98.7% 

Classes	Line Coverage	Branch Coverage	Method Coverage
<a href="#">DarstellungsInterface</a>	N/A 	N/A 	N/A 
<a href="#">FarbenLader</a>	100% 	100% 	100% 
<a href="#">FarbenWuerfel</a>	100% 	100% 	100% 
<a href="#">KameraManager</a>	94.9% 	100% 	100% 
<a href="#">M3GDarstellung</a>	96.4% 	91.3% 	92.9% 
<a href="#">M3GRubikWuerfel</a>	100% 	100% 	100% 
<a href="#">NeuzeichnerThread</a>	100% 	100% 	100% 
<a href="#">Richtungspfeile</a>	100% 	100% 	100% 
<a href="#">RotationsManager</a>	98.9% 	100% 	100% 
<a href="#">TexturenLader</a>	86.7% 	100% 	100% 
<a href="#">TexturenWuerfel</a>	100% 	100% 	100% 
<a href="#">Umgebung</a>	100% 	100% 	100% 
<a href="#">Wuerfel</a>	100% 	100% 	100% 
<a href="#">WuerfelInterface</a>	N/A 	N/A 	N/A 

**Legend:**

CCN: Cyclomatic Complexity

NCSS: Non-Commenting Source Statement

JVDC: JaVaDoc Comment

Reports generated by [Cobertura for J2ME](#).

1/28/07 10:54 PM

Abbildung 8 - Report aller Testfälle (3D View)

## Glossar

Branch Coverage	Mit Branch Coverage wird die Überdeckungsmessung etwas feiner in der Granularität in ihrer Betrachtung. Da manche Anweisungen nur unter bestimmten Bedingungen ausgeführt werden dürfen, müssen entsprechende Testfälle alle Möglichkeiten durchspielen. Branch Coverage oder Kantenüberdeckung bedient sich dabei eines anschaulichen Hilfsmittels, dem Kontrollflussgraphen, um die Kontrollstruktur des Programms darzustellen. Ein Kontrollflussgraph ist ein gerichteter Graph aus einer endlichen Menge von Knoten, welche alle für jeweils eine ausführbare Anweisung stehen., <a href="#">3</a> , <a href="#">8</a> , <a href="#">9</a>
Highscore	Eine Highscore-Tabelle bezeichnet in Computerspielen bzw. Handyspielen eine Tabelle, in der die besten Spieler nach ihren in einer Spielpartie erreichten Punkten (Highscores) absteigend sortiert dargestellt werden., <a href="#">5</a>
J2ME	Java 2 Platform, Micro Edition, abgekürzt J2ME, ist eine Umsetzung der Programmiersprache Java für so genannte "embedded consumer products" wie etwa Mobiltelefone oder PDAs. Definiert wird es in den JSR 30 und 37., <a href="#">8</a>
KToolbar	KToolbar ist eine Testumgebung für die Entwicklung von J2ME fähigen Handy-Anwendungen. Dabei simuliert KToolbar ein Handy, <a href="#">8</a>
Line Coverage	Line Coverage misst, wieviel Zeilen oder Prozent von Programmcode durch Tests ausgeführt wurden. Es wird der theoretische Ansatz der Anweisungsüberdeckung verfolgt. Ziel ist es, dass jede Anweisung einmal durchlaufen wird. Das Zusammenfassen mehrerer Anweisungen zu einer Einheit (z. B. Anweisungen innerhalb einer Schleife), die keinerlei Verzweigungen im Sinne von <b>if-else</b> Anweisungen enthält, ist bei Line Coverage möglich und wird als Block Coverage bezeichnet. Line Coverage bietet sich an, um einen ersten Überblick zu erhalten, wieviel Programmcode durch Tests durchlaufen wird., <a href="#">3</a> , <a href="#">8</a> , <a href="#">9</a>
Method Coverage	Die Methodenüberdeckung (engl. Method Coverage) zur Ermittlung der Testfälle konzentriert sich auf die einzelnen Methoden des Testobjekts. Je nach festgelegtem Ziel sind ein gewisser Anteil oder alle Methoden im Code des Testobjekts abzuarbeiten., <a href="#">8</a> , <a href="#">9</a>

Themewürfels	Statt Farben sind auf dem sog. Themewürfel sind - im Gegensatz zum Farbwürfel - einfache und kontrastreiche Texturen statt der Farben vorhanden. So wird ermöglicht, dass das Spiel auch auf einem Handy ohne Farbdisplay oder von Personen mit Rot-Grün-Sehschwäche gespielt werden kann., <a href="#">6</a>
undo	(englisch für rückgängig machen) Ist die Bezeichnung für die Funktion von Computerprogrammen, eine oder mehrere Eingaben zurückzunehmen. Je nach Komplexität der Anwendung findet sich keine Undo-Möglichkeit oder ein nur einstufiges Undo bis hin zur Rücknahme sämtlicher Arbeitsschritte., <a href="#">4</a>