

Inhaltsverzeichnis

1. Einleitung	2
2. Tests	3
2.1. Übersicht	3
2.1.1. Kategorien	3
2.2. Werkzeuge	4
2.2.1. Manuell	4
2.2.2. Automatisiert	5
2.3. Pflichtenheft	7
2.4. Protokoll	9
2.5. Nicht getestet	15
2.6. Statistik	17
2.6.1. Abdeckung	17
3. Fehler	19
3.1. Übersicht	19
3.1.1. Klassifizierung	19
3.2. Werkzeuge	19
3.2.1. Manuell	19
3.2.2. Automatisiert	20
3.3. Protokoll	21
3.4. Statistik	22
4. Änderungen	23
4.1. Protokoll	23
4.1.1. Verbessert	23
4.1.2. Nicht verbessert	24
4.1.3. Verschönert	25
5. Ausnahmen	26
5.1. Behandlung	26
5.2. Meldungen	26
6. Abschluss	27
6.1. Bewertung	27
A. Anhang	28
A.1. Screenshots	28

1. Einleitung

2. Tests

2.1. Übersicht

2.1.1. Kategorien

Wir gliedern die von uns durchgeführten Testfälle in verschiedene Kategorien:

Funktionstests

Komponententests

Zu fast jeder eigenen Komponente werden Tests durchgeführt. Wir schließen lediglich Grafik-Komponenten und reine Daten vom Testen durch Komponententests aus. Die Gründe dafür sind im Abschnitt 2.5 beschrieben. Zur Strukturierung der Tests spiegeln wir das Knot3-Projekt, welches den Programmcode enthält. D.h. zu jeder Komponente die wir testen gibt es eine Testklasse im Tests-Projekt. Eine Statistik zur Testabdeckung durch Komponententests ist unter Abschnitt 2.6.1 verfügbar.

Negativtests

Extremtests

Abnahmetests

2.2. Werkzeuge

Zur Testdurchführung helfen uns einige Werkzeuge. Wichtig bei deren Wahl waren uns diese Kriterien:

- Kostenlos
- Aktuell
- Open-Source
- In Visual Studio integrierbar
- Mit Git(-Hub) verwendbar

Eine Anleitung über die Integration und Verwendung der Werkzeuge und hilfreiche Links haben wir auf GitHub im Wiki unseres Projekts zusammengefasst. Lokal unter Visual Studio installierte Werkzeuge sind NUnit, OpenCover und ReportGenerator. Für deren Integration in Visual Studio sind NuGet Pakete verfügbar. Um die drei Werkzeuge in Visual Studio verwenden zu können, müssen sie auch aufeinander abgestimmt werden. Dazu sind Build-Skripte nötig. Unter Windows übernimmt diese Aufgabe bei uns eine einfache Stapelverarbeitungsdatei. Die „Batch“-Datei läuft beim Erstellen des Testabdeckungsberichts in Visual Studio oder lässt sich direkt ausführen.

Einerseits war es uns wichtig die Werkzeuge lokal bei jedem Entwickler verfügbar zu machen. Andererseits ist die individuelle Erstellung und Ausführung von Tests alleine sehr zeitaufwendig, weshalb wir zusätzlich Automatismen (s. Abschnitt 2.2.2) einsetzen.

2.2.1. Manuell

Werkzeuge die uns beim Schreiben und Auswerten der Tests manuell unterstützen.

NUnit , *V. 2.6.3*

NUnit ist ein Framework für Komponententests für alle .NET-Sprachen.

Internetseite: <http://www.nunit.org/>

2.2.2. Automatisiert

Zusätzlich verwenden wir serverseitige, automatisierte Dienste für Testdurchläufe und die Erstellung von Berichten, welche so ständig auf den neuesten Stand gebracht werden. Die Ergebnisse sind online abrufbar. Über bestandene und fehlgeschlagene Tests werden zudem durch einen Benachrichtigungsservice bei jeder Änderung E-Mails an die Entwickler versandt.

Visual Studio Test-Explorer , *VS-V. 12.0.21005.1*

Die Entwicklungsumgebung Visual Studio unterstützt uns beim durchführen der Tests und stellt die Ergebnisse der NUnit-Komponententests grafisch im Test-Explorer dar.

OpenCover , *V. 4.5.1923*

OpenCover ermittelt die Testabdeckung unter .NET-Sprachen ab Version 2.0. Wir nutzen es, um die Testabdeckung durch NUnit-Komponententests zu berechnen.

Internetseite: <http://opencover.codeplex.com/>

ReportGenerator , *V. 1.9.1.0*

ReportGenerator erstellt zu den von OpenCover produzierten XML-Daten einen übersichtlichen Bericht. Es sind verschiedene Formate möglich. Wir erzeugen z.B. eine HTML-Ausgabe des Berichts.

Internetseite: <http://reportgenerator.codeplex.com/>

Während unser Projekt läuft ist der automatisch erstellte Bericht über die Testabdeckung unter der Internetadresse

<http://www.knot3.de/development/coverage.php>

erreichbar.

Travis Continuous Integration (TCI)

Für private GitHub-Repositories gibt es mit TCI die Möglichkeit nach jedem Commit Tests laufen zu lassen. Führt eine Änderung zu Fehlern in bereits vorhandenen Testfällen wird dies in einer E-Mail über die Testzustände nach dem Commit an den Entwickler mitgeteilt. Der Verlauf von fehlerfreien und fehlerhafter Commits ist während der Laufzeit des Projekts unter

<https://travis-ci.org/pse-knot/knot3-code/builds>

abrufbar.

2.3. Pflichtenheft

Die Tabelle 2.1 ordnet den im Pflichtenheft vorgegebenen Testfällen einen Verweis in das Testprotokoll - wo alle Tests beschrieben werden - unter Abschnitt 2.4 zu. Da sich die Beschreibungen beim Nachspezifizieren ändern und weiter aufgliedern, erleichtert die Zuordnung das Auffinden der Pflicht-Untersuchungen. Im PDF-Dokument zum QS-Bericht führt ein Klick auf einen Bezeichner in der Spalte **Testprotokoll** direkt zu der entsprechenden Stelle im Protokoll. Während der Testphase geben die Verweise eine ständige Übersicht zum aktuellen Fortschritt und verhindern, dass bei der Vielzahl von Tests etwas vergessen wird.

Tabelle 2.1.: Pflichtenheft-Testfälle, Referenzverweise

Testfall	Verweis	
	Pflichtenheft	Testprotokoll
Funktionstests:		
Einstellen gültiger Grafikauflösungen	/PTF_10/	FT_1
Bedienen der Nutzerschnittstellen	/PTF_20/	...
Transformieren von Knoten in gültige Knoten	/PTF_20/ /PTF_70/	FT_10
Erstellen von Challenge-Leveln	/PTF_30/	...
Beenden des Programms	/PTF_50/	...
Pausieren und Beenden von Spielen	/PTF_60/	...
Manuelle Positionierung der Kamera	/PTF_80/	...
Bestehen von Challenge-Leveln	/PTF_90/	...
Speichern und Laden von Knoten	/PTF_100/	...
Einrichten und Entfernen des Programms	/PTF_120/ /PTF_130/	...

Negativtests:

Laden nicht gültiger Knoten-Dateien	/PTF_500/	...
Erstellen von Challenge-Leveln aus gleichen Knoten	/PTF_510/	...
Transformieren von Knoten in nicht gültige Knoten	/PTF_520/	...
Löschen von Standard-Leveln	/PTF_530/	...
Verhalten beim Drücken von nicht belegten Tasten	/PTF_1020/	...

Extremtests, Benchmarks:

Laden großer Knoten-Dateien	/PTF_1000/	...
Erstellen von großen Challenge-Leveln	/PTF_1010/	...
Erfassen der maximal möglichen Eingabegeschwindigkeit	/PTF_1020/	...

2.4. Protokoll

In diesem Testprotokoll sind zu jeder Testkategorie die durchgeführten Testfälle beschrieben. Über nicht durchgeführte Tests wird in Abschnitt 2.5 berichtet.

Funktionstests

FT_1 *Einstellung der Grafikauflösung.*

Die möglichen Einstellungen werden dynamisch vom Betriebssystem angefordert. D.h. die Werte, welche dem Spieler zur Auswahl stehen sind bereits vom Betriebssystem auf Gültigkeit überprüft worden (siehe: `Microsoft.Xna.Framework.Graphics.SupportedDisplayModes`).

FT_10 *Gültige Knoten-Transformationen.*

Wir definieren eine Liste möglicher Transformationen ausgehend vom Startknoten. Jede Transformation ist einzeln ausführbar.

1. Jede einzelne Kante des Startknotens ist selektierbar.
2. Mehrere Kanten (zwei, drei oder vier) des Startknotens sind selektierbar.
3. Jede einzelne Kante des Startknotens ist in jede Richtung des dreidimensionalen Raumes um einen Schritt durch direktes Anklicken und anschließendes Ziehen mit der Maus verschiebbar.
4. Jede einzelne Kante des Startknotens ist in jede Richtung des dreidimensionalen Raumes um mehrere (mindestens zehn) Schritte durch direktes Anklicken und anschließendes Ziehen mit der Maus verschiebbar.
5. Mehrere (mindestens zwei) selektierte Kanten sind um einen Schritt durch direktes Anklicken und anschließendes Ziehen mit der Maus verschiebbar.
6. Mehrere (mindestens zwei) selektierte Kanten sind um mehrere (mindestens zehn) Schritte durch direktes Anklicken und anschließendes Ziehen mit der Maus verschiebbar.
7. Jede einzelne Kante des Startknotens ist in jede Richtung des dreidimensionalen Raumes um einen Schritt durch Anklicken der Navigationspfeile und anschließendes Ziehen mit der Maus verschiebbar.
8. Jede einzelne Kante des Startknotens ist in jede Richtung des dreidimensionalen Raumes um mehrere (mindestens zehn) Schritte durch Anklicken der Navigationspfeile und anschließendes Ziehen mit der Maus verschiebbar.

9. Mehrere (mindestens zwei) selektierte Kanten sind um einen Schritt durch Anklicken der Navigationspfeile und anschließendes Ziehen mit der Maus verschiebbar.
10. Mehrere (mindestens zwei) selektierte Kanten sind um mehrere (mindestens zehn) Schritte durch Anklicken der Navigationspfeile und anschließendes Ziehen mit der Maus verschiebbar.
11. Der in ?? abgebildete, Knoten „Schlaufe“ ist erstellbar.
12. Der in ?? abgebildete, Knoten „Überleger“ ist erstellbar.
13. Jede einzelne Kante des Startknotens lässt sich nach ihrer Verschiebung in die vorige Position durch direktes Anklicken und anschließendes Ziehen zurücksetzen.
14. Jede einzelne Kante des Startknotens lässt sich nach ihrer Verschiebung in die vorige Position durch Anklicken des „Undo“-Buttons zurücksetzen.
15. Jede einzelne Kante des Startknotens lässt sich nach ihrer Verschiebung in die vorige Position durch Anklicken des „Undo“-Buttons zurücksetzen und der „Redo“-Button macht die Aktion des „Undo“-Buttons rückgängig.

FT_020 *Erstellen von Challenge-Leveln*

1. Im Hauptmenü auf den Text „NEW Creative“ klicken.
2. Im Creative-Menü auf den Text „NEW Challenge“ klicken.
3. Im Challenge-Konstruktions-Menü in der linken Liste einen Zielknoten auswählen.
4. In der rechten Liste einen Startknoten auswählen.
5. Im Eingabefeld einen Namen für die Challenge eingeben und mit der „ENTER“-Taste bestätigen.

Komponententests

Beschreibungen zu den getesteten Komponenten.

Negativtests

Extremtests

Abnahmetests

2.5. Nicht getestet

Hier beschreiben wir nicht getestetes Verhalten und begründen im konkreten Fall unsere Entscheidung einen Test nicht durchzuführen.

Funktionstests

Komponententests

Von den Klassen, die wir für das Unit-Testing in Betracht gezogen haben, mussten wir diejenigen ausschließen, die für einen entscheidenden Teil ihrer Funktionalität eine oder mehrere Instanzen der Klassen Game, GraphicsDeviceManager, GraphicsDevice und ContentManager benötigen. Das bedeutet, dass sie Instanzen dieser Klassen entweder im Konstruktur erstellen, im Konstruktur als Parameter erwarten, dass sie teilweise Wrapper um diese Klassen sind oder dass ihre Funktionalität sich auf einige wenige Methoden beschränkt, die mit diesen Instanzen arbeiten.

Dazu gehören einerseits alle von IRenderEffect erbenenden Klassen wegen der intensiven Nutzung von GraphicsDevice und GraphicsDeviceManager sowie teilweise von Instanzen der Klasse Effect, das den Zugriff auf Shader kapselt und ebenfalls von GraphicsDevice und ContentManager abhängt.

Andererseits gehören dazu auch die GameModels und davon erbenende Klassen, weil diese eine Instanz von Model enthalten, das über einen ContentManager geladen werden muss und damit ein GraphicsDevice benötigen. Auch die InputHandler, deren hauptsächliche Funktion es ist, in bestimmten Methoden, die eventbasiert aufgerufen werden, Listen von GameModels zu erstellen, sind damit von ContentManager und vom GraphicsDevice abhängig.

Negativtests

Extremtests

Abnahmetests

2.6. Statistik

2.6.1. Abdeckung

Auf den folgenden Seiten steht der aus den Daten von OpenCover generierte Bericht zur Testabdeckung durch Komponententests.

Summary

Generated on: 18.02.2014 - 14:16:32
Parser: OpenCoverParser
Assemblies: 1
Classes: 41
Files: 40
Coverage: 55.9%
Covered lines: 1418
Uncovered lines: 1115
Coverable lines: 2533
Total lines: 8365

Assemblies

Knot3	55.9%
Knot3.Audio.AudioManager	45.2%
Knot3.Audio.LoopPlaylist	0%
Knot3.Audio.OggVorbisFile	23.5%
Knot3.Audio.SoundEffectFile	0%
Knot3.Core.Angles3	100%
Knot3.Core.BooleanOptionInfo	0%
Knot3.Core.Camera	65.3%
Knot3.Core.ConfigFile	100%
Knot3.Core.DisplayLayer	98.1%
Knot3.Core.DistinctOptionInfo	0%
Knot3.Core.FloatOptionInfo	0%
Knot3.Core.KeyOptionInfo	0%
Knot3.Core.Localizer	0%
Knot3.Core.OptionInfo	0%
Knot3.Core.Options	81.2%
Knot3.Core.World	15.9%
Knot3.Data.Challenge	0%
Knot3.Data.ChallengeFileIO	0%
Knot3.Data.ChallengeMetaData	0%
Knot3.Data.CircleEntry‘1	92.3%
Knot3.Data.CircleExtensions	100%
Knot3.Data.Direction	98.6%
Knot3.Data.Edge	95%
Knot3.Data.Knot	84.8%
Knot3.Data.KnotFileIO	22.2%
Knot3.Data.KnotMetaData	64%
Knot3.Data.KnotStringIO	41.1%
Knot3.Data.Node	73.7%
Knot3.Data.NodeMap	88.5%
Knot3.Data.PrinterIO	0%
Knot3.Data.RectangleMap	0%
Knot3.Data.ZipHelper	0%
Knot3.Platform.SystemInfo	88.6%
Knot3.Utilities.BoundingCylinder	0%
Knot3.Utilities.FileIndex	0%
Knot3.Utilities.FileUtility	36.1%
Knot3.Utilities.IniFile	81.1%
Knot3.Utilities.RayExtensions	0%
Knot3.Utilities.SavegameLoader‘2	0%
Knot3.Widgets.Bounds	83.5%
Knot3.Widgets.ScreenPoint	50.8%

3. Fehler

3.1. Übersicht

3.1.1. Klassifizierung

Programmfehler Fehler im Programm.

Grafikfehler Fehlerhafte grafische Darstellung.

Fehlend Fehlender Bestandteil.

3.2. Werkzeuge

Bei der Fehlersuche unterstützen uns mehrere Programme. Je nach Fehlerklasse (s. 3.1.1) sind verschiedene Werkzeuge hilfreich.

3.2.1. Manuell

FastStone Capture , V. 7.7

FastStone Capture erstellt Bildschirmaufnahmen. Damit lassen sich Screenshots und Videos von mehreren Fenstern machen. In den Videos werden auch Benutzerinteraktionen eingezeichnet. Gerade bei Fehlern, die sich durch grafisches Fehlverhalten äußern und um diese zu dokumentieren, kommt dieses Werkzeug zum Einsatz. Die Screenshots helfen bei der Fehlerbeschreibung. Zudem lassen sich die Videos - deren Größe wenige MB beträgt - einfach ins GIF-Format konvertieren. Das ist besonders hilfreich, da sich bis jetzt unter GitHub nur GIF-Animationen in die textuelle Beschreibung direkt einfügen lassen. Im Gegensatz zu den anderen Werkzeugen ist diese Software Shareware.

Internetseite: <http://www.faststone.org>

GitHub-Ticket-System

Das durch GitHub bereit gestellte Ticket-System nutzen wir zur Fehlerverfolgung. Sämtliche Probleme sind dort unter

<https://github.com/pse-knot/pse-knot/issues>

aufgelistet.

3.2.2. Automatisiert

Gendarme

Gendarme durchsucht anhand von Regeln .NET-Code und gibt einen Fehlerbericht aus. Das Werkzeug kontrolliert u. A.:

- Code-Style
- Code-Conventions
- Änderungen, welche die Performance verbessern
- ...

Internetseite: <http://www.mono-project.com/Gendarme>

Während der Laufzeit des Projekts liegt der Gendarme-Bericht unter

<http://www.knot3.de/development/gendarme.html>

vor.

3.3. Protokoll

3.4. Statistik

4. Änderungen

4.1. Protokoll

4.1.1. Verbessert

4.1.2. Nicht verbessert

4.1.3. Verschönert

5. Ausnahmen

5.1. Behandlung

5.2. Meldungen

6. Abschluss

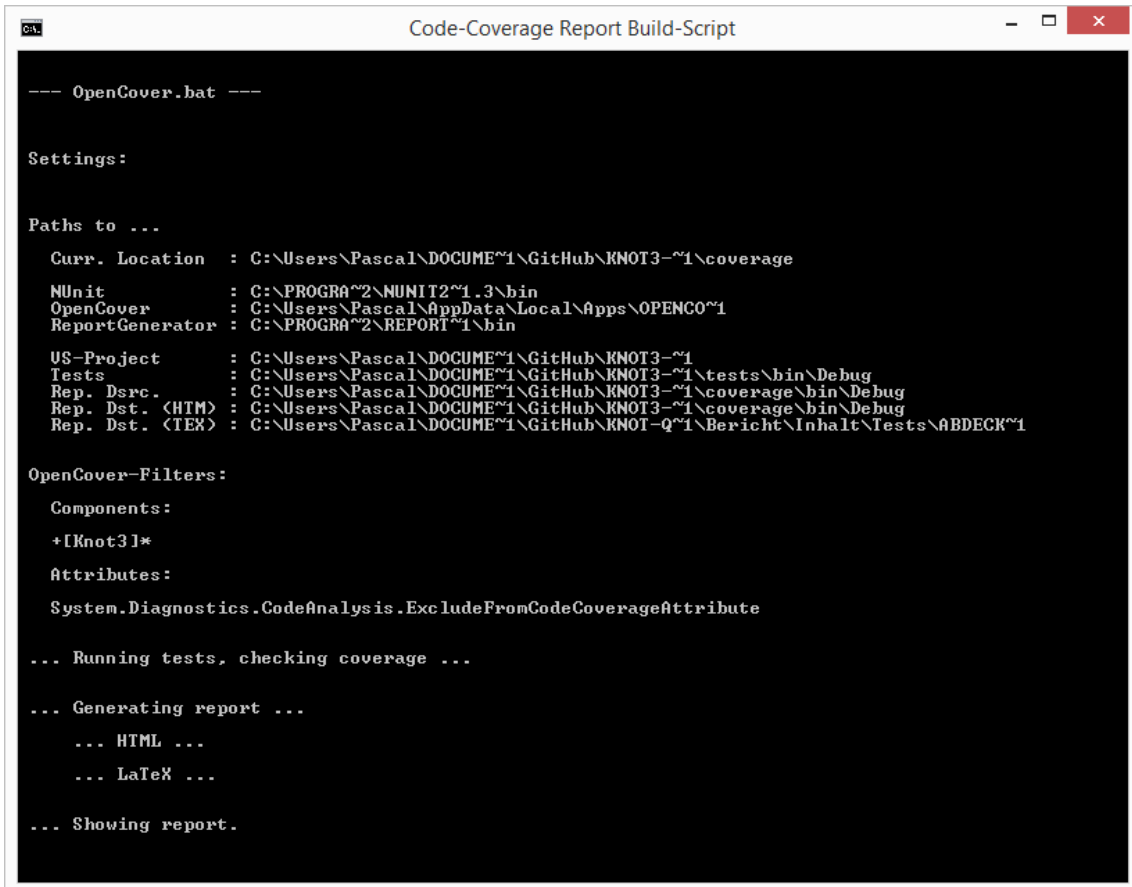
6.1. Bewertung

A. Anhang

A.1. Screenshots

Grafikfehler

Werkzeuge



```
Code-Coverage Report Build-Script

--- OpenCover.bat ---

Settings:

Paths to ...

  Curr. Location   : C:\Users\Pascal\DOCUME~1\GitHub\KNOT3-~1\coverage
  NUnit            : C:\PROGRA~2\NUNIT2~1.3\bin
  OpenCover        : C:\Users\Pascal\AppData\Local\Apps\OPENC0~1
  ReportGenerator  : C:\PROGRA~2\REPORT~1\bin

  US-Project       : C:\Users\Pascal\DOCUME~1\GitHub\KNOT3-~1
  Tests            : C:\Users\Pascal\DOCUME~1\GitHub\KNOT3-~1\tests\bin\Debug
  Rep. Dsrc.       : C:\Users\Pascal\DOCUME~1\GitHub\KNOT3-~1\coverage\bin\Debug
  Rep. Dst. <HTM>  : C:\Users\Pascal\DOCUME~1\GitHub\KNOT3-~1\coverage\bin\Debug
  Rep. Dst. <TEX>  : C:\Users\Pascal\DOCUME~1\GitHub\KNOT-~1\Bericht\Inhalt\Tests\ABDECK~1

OpenCover-Filters:

  Components:

  +[Knot3]*

  Attributes:

  System.Diagnostics.CodeAnalysis.ExcludeFromCodeCoverageAttribute

... Running tests, checking coverage ...

... Generating report ...

  ... HTML ...

  ... LaTeX ...

... Showing report.
```

Abbildung A.1.: Build-Batch für die Erstellung des Testabdeckungs-Berichts.