Inhaltsverzeichnis

1. Einleitung

2. Tests

2.1. Übersicht

Die von uns durchgeführten Tests gliedern sich in:

Funktionstests

Komponententests

Negativtests

Extremtests

Abnahmetests

Zudem beschreiben wir im Abschnitt 2.4.6 was wir nicht getestet haben und aus welchen Gründen.

2.2. Werkzeuge

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

- Für Studenten kostenlos, da es hier um ein studentisches Projekt geht.
- Aktuelle Open-Source-Software.
- Lokale Integrierbarkeit in unsere Entwicklungsumgebung.
- Verwendbarkeit zusammen Git(-Hub).

2.2.1. Manuelle Unterstützung

Eine Anleitung über die Integration und Verwendung der Werkzeuge und hilfreiche Links ist auf GitHub im Wiki unseres Projekts verfügbar. Lokal unter Visual Studio installierte Werkzeuge sind:

NUnit , *V. 2.6.3*

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

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

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/

Für die Integration in Visual Studio sind NuGet Pakete für NUnit, OpenCover und ReportGenerator verfügbar.
Um die drei Werkzeuge in Visual Studio verwenden zu können, müssen sie zunächst aufeinander abgestimmt werden. Dazu sind Build-Skripte nötig. Unter Windows übernimmt diese Aufgabe bei uns eine einfache Stapelverarbeitungsdatei (Batch-Datei/.bat-Dateiendung).

Einerseits ist es uns wichtig die Werkzeuge lokal bei jedem Entwickler verfügbar zu machen. Andererseits ist die individuelle Erstellung und Ausführung von Tests alleine noch zu zeitaufwendig.

2.2.2. Automatisierte Tests

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 (s. u.,). Über bestandene und fehlgeschlagene Tests werden zudem durch einen Benachrichtigungsservice bei jeder Änderung E-Mails an die Entwickler versandt.

OpenCover und ein eigener HTML-Generator

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-Testfälle, Referenzverweise

Die Tabelle 2.1 ordnet den im Pflichtenheft vorspezifizierten Testfällen einen Verweis in das Testprotokoll - wo alle Tests beschrieben werden - zu. Da sich deren Beschreibungen beim Nachspezifizieren ändern und weiter aufgliedern, erleichtert die Zuordnung das Auffinden der Pflicht-Untersuchungen: ein Klick auf einen Bezeichner in der Spalte **Testprotokoll** führt direkt zu der entsprechenden Stelle des Protokolls. Zudem geben die Verweise während der Testphase eine ständige Übersicht zum aktuellen Fortschritt und verhindern, dass bei einer 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-Spielen	/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. Protokolle

2.4.1. 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 *Eine neue Challenge aus zwei im Creative-Mode erzeugten Knoten erstellen.*

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

2.4.2. Komponententests

Wir führen für fast jede Komponente Tests durch. Davon ausgenommen sind:

Grafik-Komponenten

Daten

Zur Strukturierung der Test spiegeln wir das 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 verfügbar (siehe 2.5.1).

2.4.3. Negativtests

2.4.4. Extremtests, Benchmarks

2.4.5. Abnahmetests

2.4.6. Nicht getestet

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 Konstuktor als Paramater 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 erbenden 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 erbende 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.

2.5. Statistik

2.5.1. Testabdeckung

Der automatisch generierte Bericht (s. **??**) zur Testabdeckung durch Komponententests.

3. Programmfehler

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. Manuelle Unterstützung

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 Sharewa-
re.

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. Automatisierte Überprüfung

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. Statistik

4. Änderungen

4.1. Protokoll

4.1.1. Behobene Probleme

4.1.2. Nicht behobene Probleme

4.1.3. Verschönerungen

5. Ausnahmen

5.1. Behandlungen

5.2. Meldungen

6. Abschluss

6.1. Bewertung

A. Anhang

A.1. Vollständige Programmfehlerübersicht

13.02.2014 - 17:22

KnotStringIO

Der Setter für das Content-Property. Es wurde auf einzelne Zeichen anstatt auf mehrere Zeilen überprüft.

A.2. Programmaufnahmen