

Qualitätssicherungsbericht

(V. 1.0)

PSE WS 2013/14



Auftraggeber:
Karlsruher Institut für Technologie
Institut für Betriebs- und Dialogsysteme
Prof. Dr.-Ing. C. Dachsbacher

Betreuer:
Dipl.-Inf. Thorsten Schmidt
Dipl.-Inform. M. Retzlaff

Auftragnehmer:
Tobias Schulz, Maximilian Reuter, Pascal Knodel,
Gerd Augsburg, Christina Erler, Daniel Warzel

9. März 2014

Inhaltsverzeichnis

1. Einleitung	3
2. Tests	4
2.1. Übersicht	4
2.1.1. Kategorien	4
2.2. Werkzeuge	6
2.2.1. Manuell	6
2.2.2. Automatisiert	7
2.3. Pflichtenheft	9
2.4. Protokoll	11
2.5. Nicht getestet	23
2.6. Statistik	24
2.6.1. Abdeckung	24
3. Fehler	27
3.1. Übersicht	27
3.1.1. Klassifizierung	27
3.2. Werkzeuge	28
3.2.1. Manuell	28
3.2.2. Automatisiert	29
3.3. Protokoll	30
3.4. Statistik	33
4. Änderungen	34
4.1. Protokoll	34
4.1.1. Geändert	34
4.1.2. Nicht geändert	37
4.1.3. Nicht verschönert	38
5. Ausnahmen	39
5.1. Behandlung	39
5.2. Meldungen	39
6. Schluss	40
6.1. Bewertung	40
A. Anhang	41
A.1. Aufnahmen	41
A.1.1. Testknoten	42

1. Einleitung

In diesem Dokument beschreiben wir, was in der Qualitätssicherungsphase von uns erarbeitet wurde. Aus diesen Daten ziehen wir Rückschlüsse, welche qualitativen Anforderungen wir gezielt untersucht und getestet haben und geben eine Bewertung der Qualität des Spiels an.

Da in der Implementierungsphase aus zeitlichen Gründen nicht alle Pflichtenheft-Spezifikationen umgesetzt wurden, haben wir Ausstehendes in der QS-Phase nachgearbeitet. U. A. die Knoten-Vorschau beim Transformieren gehört zu den wichtigeren Nachträgen. Änderungen und Ergänzungen, welche nicht oder in den vorigen Phasen anders spezifiziert waren, werden in diesem Bericht dokumentiert. Das betrifft z.B. zusätzliches Text-Material, welches wir zusammen mit dem Spiel ausliefern, um das Spiel an sich, die Steuerung oder Inhalte für die Spieler zu beschreiben. Neben den Angaben zu den durchgeführten Tätigkeiten und Änderungen enthält der Bericht auch die Dinge, deren Umsetzung nicht möglich war.

Zur Unterstützung der Zusammenarbeit und als Kommunikationsmittel verwendeten wir das „Issues“-Ticket-System von GitHub. Dort sind alle von uns gefundenen Fehler erfasst. Deren Korrektur stand neben dem Testen in dieser Phase im Vordergrund. Der QS-Bericht enthält Protokolle zu den Fehlern, Tests und Änderungen, die im verbliebenen zeitlichen Rahmen möglich waren.

Die neueste Version von Knot3 mit allen Änderungen die während der Qualitätssicherung noch mit einflossen ist unter der Internetadresse

¶ <http://www.knot3.de/download/>

oder auf GitHub unter

¶ <https://github.com/pse-knot/knot3-code/releases>

verfügbar.

2. Tests

2.1. Übersicht

2.1.1. Kategorien

Wir gliedern die von uns durchgeführten Testfälle in verschiedene Kategorien:

Funktionstests

In dieser Kategorie testen wir die Spielfunktionen. Dabei handelt es sich um verbale Beschreibungen von Vorgehen, deren Durchführbarkeit wir mehrfach erfolgreich getestet haben. Wir gewährleisten, dass diese aufgelistete Funktionalität durchführbar ist. Das sind Funktionen und Abläufe, welche für die Spielbarkeit von Knot3 benötigt werden. Die Liste steht im entsprechenden Abschnitt im Testprotokoll, ab § S. 12.

Komponententests

Das sind Tests zu einzelnen C#-Komponenten unseres Projekts. Da verschiedene Komponententests oft sehr ähnlich in der Umsetzung sind (Einsetzen von Beispiel-objekten/-werten) geben wir im entsprechenden Abschnitt im Testprotokoll, ab § S. 18, eine Zusammenfassung an, anstatt jeden Komponententest einzeln zu beschreiben. Zu fast jeder Komponente führen wir Tests durch, außer an reinen Grafik- oder Daten-Komponenten. Die Gründe dafür sind im Abschnitt § 2.5, ab S. 23 nochmals im Detail erklärt. Die Statistik zur Testabdeckung durch Komponententests ist unter Abschnitt § 2.6.1, ab S. 24 verfügbar.

Negativtests

Die Negativtests prüfen, ob das Spiel eine (falsche) Eingabe oder Bedienung, welche nicht den Anforderungen an die Anwendung entsprechen erwartungsgemäß abweist. Siehe dazu im entsprechenden Abschnitt im Testprotokoll, ab ↗ S. 19.

Extremtests

Bei den Extremtests prüfen wir, ob das Spiel größere Datenmengen problemlos verarbeitet und wo die oberen Schranken liegen. Zudem führen wir einfaches Benchmarking durch. Wir messen den Zeitbedarf kritischer Funktionen und schauen nach möglichen Flaschenhälsern. Die Testergebnisse finden sich im entsprechenden Abschnitt im Testprotokoll ab ↗ S. 20.

Abnahmetests

Hierbei lassen wir menschliche Tester unser Spiel spielen. Deren Kommentare und Kritiken zu Knot3 sind im entsprechenden Abschnitt im Testprotokoll ab ↗ S. 22 beschrieben.

2.2. Werkzeuge

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

- Kostenlos (für studentische Projekte)
- 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 - der ↗ Screenshot im Anhang zeigt den Ablauf.

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. Deshalb setzen wir zusätzlich Automatismen ein, ↗ siehe Abschnitt 2.2.2, ab S. 7.

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/>

MonoGame-SDL2 , *Aktuelle Git-Version*

Ein Branch/Fork von MonoGame, der eine saubere quelloffene Neuimplementierung von XNA auf Basis von SDL2 ist und eine erweiterte API besitzt.

Internetseite: <https://github.com/flibitijibibo/MonoGame>

Die von uns verwendete Version: <https://github.com/tobiasschulz/MonoGame>

MonoDevelop / Xamarin Studio , *V. 4.2.2*

Eine integrierte Entwicklungsumgebung, die im Rahmen des Mono-Projektes als Open-Source entwickelt wird und unter Linux und Windows verwendet werden kann. Sie wird von uns zur Kompilierung von Knot3 mit MonoGame statt mit XNA genutzt und ist bei der Entwicklung unter Linux ein vollwertiger Ersatz für Visual Studio. Die Linux-Version heißt MonoDevelop und die Windows-Version wird von der Firma Xamarin, die das Mono-Projekt leitet, unter dem Namen Xamarin Studio vermarktet.

Internetseite: <http://monodevelop.com/download>

Knot3 Visual Tests

Ein Tool, das die Zeit misst, die zur Darstellung von Konten benötigt wird. Dabei kann die Anzahl der Kanten des zu generierenden Knotens eingestellt werden und es werden die Frames pro Sekunde sowie die Zeit, die zum Zeichnen benötigt wird, angezeigt. Das Tool ist nur mit MonoGame und MonoDevelop und nicht mit XNA kompatibel.

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.

MonoDevelop NUnit Add-in , V. 4.0.12

Die Entwicklungsumgebung MonoDevelop / Xamarin Studio unterstützt uns beim Durchführen der Tests und stellt die Ergebnisse der NUnit-Komponententests grafisch im mit MonoDevelop mitgelieferten NUnit-Add-in 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/>

Eigenentwicklung zum Generieren eines Coverage-Reports unter Mono, *Aktuelle Git-Version*

Während unser Projekt läuft ist ein automatisch von unserem Server erstellter Bericht über die Testabdeckung unter der Internetadresse

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

erreichbar. Dieser wird von einer Eigenentwicklung generiert, dessen Quellcode in einem GitHub-Repository unter

<https://github.com/knot3/CSharpCoverage>

bereitgestellt wird.

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 vorspezifizierten Testfällen einen Verweis in das Testprotokoll - wo alle Tests beschrieben werden - ⁴ unter Abschnitt 2.4, ab S. 11 zu. Da sich die Beschreibungen beim Nachspezifizieren ändern oder weiter aufgliedern, erleichtert die Zuordnung das Auffinden der Pflichtenheft-Tests. Im PDF-Dokument zum QS-Bericht führt ein Klick auf einen Bezeichner in der Spalte **Testprotokoll** 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
Funktionstests:	Pflichtenheft Testprotokoll
Einstellen gültiger Grafikauflösungen	/PTF_10/ FT_1, S. 23
Spiel-Modi starten	/PTF_20/ FT_100, S. 16
Transformieren von Knoten in gültige Knoten	/PTF_20/, /PTF_70/ FT_10, S. 12
Erstellen von Challenge-Leveln	/PTF_30/ FT_20, S. 13
Beenden des Programms	/PTF_50/ FT_40, S. 13
Pausieren und Beenden von Spielen	/PTF_60/ FT_50, S. 13
Manuelle Positionierung der Kamera	/PTF_80/ FT_80, S. 15
Bestehen von Challenge-Leveln	/PTF_90/ FT_60,, S. 14
Speichern und Laden von Knoten	/PTF_100/ FT_70, S. 15
Einrichten und Entfernen des Programms	/PTF_120/, /PTF_130/ FT_90, S. 16

Negativtests:

Laden nicht gültiger Knoten-Dateien	/PTF_500/	NT_10, S. 19
Erstellen von Challenge-Leveln aus gleichen Knoten	/PTF_510/	NT_20, S. 19
Transformieren von Knoten in nicht gültige Knoten	/PTF_520/	NT_30, S. 19
Löschen von Standard-Leveln	/PTF_530/	NT_40, S. 19
Verhalten beim Drücken von nicht belegten Tasten	/PTF_1020/	NT_50, S. 19

Extremtests, Benchmarks:

Laden großer Knoten-Dateien	/PTF_1000/	ET_1, S. 20
Erstellen von großen Challenge-Leveln	/PTF_1010/	ET_1, S. 20

2.4. Protokoll

In diesem Abschnitt sind zu jeder Testkategorie die durchgeführten Testfälle beschrieben. Über nicht durchgeführte Tests wird unter \triangleleft 2.5 berichtet.

- \triangleleft Funktionen, S. 12
- \triangleleft Komponenten, S. 18
- \triangleleft Negativ-Fälle, S. 19
- \triangleleft Extremfälle, S. 20
- \triangleleft Spielbarkeit, S. 22

Funktionstests

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. Jede einzelne Kante des Startknotens lässt sich nach ihrer Verschiebung in die vorige Position durch direktes Anklicken und anschließendes Ziehen zurücksetzen.
 12. Jede einzelne Kante des Startknotens lässt sich nach ihrer Verschiebung in die vorige Position durch Anklicken des „Undo“-Buttons zurücksetzen.
- ...

13. 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_20 Erstellen von Challenge-Leveln.

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

FT_30 Nachbaubare Knoten-Beispiele.

Eine Sammlung von Beispiel-Knoten zum Nachbauen. Jeder Knoten deckt einen im Spielverlauf immer wieder-aufzutretenden Modellierungsfall einmalig ab.

1. ↗ „Überleger“
2. ↗ „Schlaufe“

FT_40 Beenden des Programms.

Ein Klick auf den „Exit“-Button im Hauptmenü beendet das Programm.

FT_50 Pausieren und Beenden von Spielen.

In beiden Spielmodi besteht die Möglichkeit ein Spiel zu pausieren und zu beenden.

Pausieren eines laufenden Spiels:

1. Durch Drücken der „ESC“ -Taste im laufenden Spiel öffnet sich das Pausemenü. Im Challenge-Mode wird die Challenge-Zeit hierbei pausiert.
2. Durch ein Klick auf den Text „Back to Game“ im Pausemenü wird dieses Menü geschlossen und das Spiel fortgesetzt. Im Challenge-Mode läuft nach dem Schließen des Pausemenüs die Challenge-Zeit weiter.

Beenden eines laufenden Spiels im Challenge-Mode:

1. Durch Drücken der „ESC“ -Taste im laufendem Spiel öffnet sich das Pausenmenü.
2. Durch einen Klick auf den Text „Abort Challenge“ schließt sich die laufende Challenge und öffnet das Hauptmenü.

Beenden eines laufenden Spiels im Creative-Mode:

1. Durch Drücken der „ESC“ -Taste im laufendem Spiel öffnet sich das Pausenmenü.
2. Durch einen Klick auf den Text „Save and Exit“ wird der Knoten gespeichert:
 - Fall 1: Ist bereits ein Dateiname vorhanden, wird dieser beim Speichern verwendet.
 - Fall 2: Ist noch kein Dateiname vorhanden, öffnet sich der „Save As“-Dialog und fordert den Spieler auf einen Namen einzugeben. Der Knoten wird nach Bestätigen dieses Dialogs gespeichert und der Spieler gelangt ins Hauptmenü.
3. Durch einen Klick auf den Text „Discard Changes and Exit“ gelangt der Spieler ins Hauptmenü.

FT_60 Bestehen von Challenge-Leveln.

Nachdem der Spieler die letzte Transformation zur Beendigung der Challenge getätigkt hat, reagiert das Spiel folgendermaßen:

1. Die Challenge-Zeit des Spielers wird gestoppt.
2. Es öffnet sich ein Dialog, welcher den Spieler auffordert seinen Spielernamen einzugeben.
3. Nachdem der Spieler den Spielernamen mit der „ENTER“ -Taste bestätigt hat, wird die Highscore-Liste geöffnet.
4. In der Highscore-Liste kann der Spieler die 10 besten Highscore-Einträge sehen. Wenn die Challenge-Zeit des Spielers gereicht hat, besitzt dieser auch einen Highscore-Eintrag.
5. Mit Hilfe der zwei Buttons („Restart challenge“ und „Return to menu“) kann der Spieler die Challenge noch einmal spielen oder zum Hauptmenü zurückkehren.

FT_70 Speichern und Laden von Knoten.

Hat der Spieler im Creative-Mode einen Knoten erstellt, so kann er diesen abspeichern und wiederum laden. Dazu muss man folgende Dinge tun:

1. Durch Drücken der „ESC“ -Taste öffnet sich das Pausemenü.
2. Im Pausemenü kann man den Knoten auf unterschiedliche Art und Weise speichern:
 - Durch einen Klick auf den Text „Save“ . wird man aufgefordert den Knotennamen einzugeben, welchen man mit der „ENTER“ -Taste bestätigt. Hat der Knoten bereits einen Knotennamen, so wird man nicht mehr aufgefordert diesen einzugeben. Daraufhin wird der Knoten unter diesem Namen gespeichert.
 - Durch einen Klick auf den Text „Save As“ wird man aufgefordert den Knotennamen einzugeben, welchen man mit der „ENTER“ -Taste bestätigt. Daraufhin wird der Knoten unter diesem Namen gespeichert.
 - Durch einen Klick auf den Text „Save and Exit“ wird man aufgefordert den Knotennamen einzugeben, welchen man mit der „ENTER“ -Taste bestätigt. Hat der Knoten bereits einen Knotennamen, so wird man nicht mehr aufgefordert diesen einzugeben. Nach der Bestätigung wird der Knoten gespeichert und das Spiel kehrt zurück zum Hauptmenü.
3. Im Hauptmenü auf den Text „Creative“ klicken.
4. Durch einen Klick auf den Text „LOAD Knot“ im Creative-Menü öffnet sich das Knoten-Lademenü.
5. Im Knoten-Lademenü kann man aus der linken Liste den zuvor abgespeicherten Knoten auswählen und mit dem „Load“ -Button laden.

FT_80 Manuelle Positionierung der Kamera.

Mit den folgenden Tastatureingaben kann der Spieler die Kamera manuell bewegen. Die Tastatureingaben sind der Standardtastaturbelegung entnommen (siehe Spielanleitung/Kamerabewegung).

- Mit den „WASD“ Tasten bewegt der Spieler die Kamera nach oben/unten/rechts/links.
- Mit Hilfe der Tasten „R“ und „F“ bewegt der Spieler die Kamera in der Ebene nach vorne und hinten.
- Der Spieler zoomt mit den Tasten „Q“ und „E“ (alternativ mit dem Mausrad) herein- und heraus.
- Der Spieler rotiert die Kamera um eine Kante des Knotens, indem er sie mit der rechten Maustaste auswählt und mit den Pfeiltasten (alternativ durch gedrückt halten der rechten Maustaste) um die Kante rotiert.

- Mittels der „Space“ -Taste springt der Mittelpunkt der Kamera auf den Mittelpunkt der selektierten Kante.
- Mit der linken „Alt“ -Taste wird die Kamera frei gegeben. Mit der Maus schaut sich der Spieler um. Durch erneutes Klicken der linken „Alt“ -Taste rastet die Kamera wieder ein.
- Durch drücken der „ENTER“ -Taste setzt der Spieler die Kamera zurück.

FT_90 Einrichten und Entfernen des Programms

Hinweis: Es gibt in der Endversion von Knot3 keine automatische Installation/Deinstallation, .

1. Das Archiv in dem sich alle für das Spiel relevanten Dateien befinden lässt sich auf dem Zielsystem entpacken.
2. Durch einen Doppel-Klick auf die ausführbare Datei „Knot3.exe“ startet das Spiel erstmalig im Fenstermodus und das Hauptmenü wird angezeigt. Dabei wird auf dem Zielsystem auch ein Ordner für Einstellungen und Spielspeicherstände angelegt.
3. Die Deinstallation erfolgt manuell. D.h. alle zu Knot3 gehörigen Ordner sind vom System zu löschen.

FT_100 Spiel-Modi starten

Creative-Mode:

1. Durch einen Klick auf den Text „Creative“ im Hauptmenü öffnet sich das Creative-Menü.
2. Durch einen Klick auf den Text „NEW Knot“ startet der Creative-Mode zum Erstellen eines neuen Knotens.
3. Als Start-Knoten wird ein Quadrat angezeigt.

Challenge-Mode:

1. Durch einen Klick auf den Text „Challenge“ im Hauptmenü öffnet sich das Challenge-Menü.
2. Im Challenge-Menü wird in der Challenge-Liste eine Challenge ausgewählt und durch einen Klick auf den Start-Button gestartet.

3. Auf der linken Seite des Bildschirms wird der Referenzknoten, auf der rechten Seite der zu bearbeitende Knoten angezeigt.

Komponententests

Da Komponententests zum Testen bestimmter funktionalen Einheiten des Programms gedacht sind, können wir einen großen Teil unseres Spiels so nicht testen, da dieser auf Benutzerinteraktionen wartet oder zum Anzeigen grafischer Oberflächen gebraucht wird. Mehr dazu in Abschnitt \triangleleft 2.5 „Nicht Getestet“.

Die von uns getesteten Klassen umfassen die Daten-Klassen, die sich um die Datenstrukturen unserer Knoten kümmern, sowie auch die Klassen, die sich mit dem Speichern und Laden von Dateien wie zum Beispiel den Einstellungen befassen. Des weiteren testen wir auch noch unsere mathematischen Klassen.

Um bestimmte Tests durchführen zu können haben wir uns sogenannte Mock-Objekte erstellt. Das sind meist leere oder mit nur sehr einfachen Daten gefüllte Objekte, die von anderen Klassen vorausgesetzt werden um diese testen zu können. Wir haben zum Beispiel einen FakeScreen angelegt, der nur dafür da ist, dass die Bounds und ScreenPoint Klassen getestet werden können.

Für das Testen unserer Datenstrukturen haben wir uns einen Knot-Generator erstellt, der mit angegeben Parametern quadratische Knoten erstellen kann, die wir durch unsere diversen Tests für Knoten laufen lassen können.

Im Zuge unserer Komponententests ist uns so unter anderem aufgefallen, dass unsere Knoten-Laderoutine nur Knoten mit RGBA Farbwerten einlesen kann, nicht wie von uns gefordert auch Knoten mit RGB Farbwerten. Da sich der Funktionale Teil unseres Projektes auf ein paar Dateisystem-Interaktionen sowie das Laden und Erstellen der Datenstruktur beschränkt und der Großteil unserer Interaktionen und Manipulationen dieser Strukturen im Laufenden Spiel stattfindet, hält sich die Effizienz der Komponententests in Grenzen. Daher führen wir auch nutzerbasierte Tests, wie in Abschnitt \triangleleft 2.4 beschrieben durch, um die Qualität unseres Programms sicherstellen zu können.

Negativtests

Problematische Eingaben und Spielsituationen werden hier explizit getestet.

NT_10 *Laden nicht gültiger Knoten-Daten.*

Es ist nicht möglich ungültige Knoten-Daten zu laden. Der Algorithmus lässt dies nicht zu. Nur gültige Daten, die dem von uns spezifizierten Format entsprechen werden in der Auswahlliste angezeigt.

NT_20 *Erstellen von Challenge-Leveln aus gleichen Knoten.*

Das Erstellen einer Challenge mit gleichen Knoten ist nicht möglich. Wählt der Spieler beim Erstellen einer Challenge zwei gleiche Knoten aus, so kann er nicht auf den „Create!“ -Button drücken.

NT_30 *Transformieren von Knoten in nicht gültige Knoten.*

Da nur gültige Transformationen durchführbar sind, ist es nicht möglich einen ungültigen Knoten mit Hilfe von Transformationen durch die dem Spieler zur Verfügung gestellten Eingabemethoden zu erstellen. Es ist nicht möglich Kanten an Kanten zu ziehen und so Überlappungen zu erzeugen.

NT_40 *Löschen von Standard-Leveln.*

Löschen von Spielständen wird über das Löschen der Dateien im persönlichen Ordner vorgenommen. Unter Windows sind die Standard-Level nicht im persönlichem Ordner und können somit dort auch nicht gelöscht werden. Unter Linux werden nicht mehr vorhandene Standard-Level beim Starten neu in den persönlichen Ordner kopiert.

NT_50 *Verhalten beim Drücken von nicht belegten Tasten.*

Drückt der Spieler nicht belegte Tasten so passiert nichts. Das Spiel läuft ohne Probleme weiter. Es verhält sich so in allen Spielsituationen.

Extremtests

Zunächst haben wir eine Hilfsklasse für einfaches Benchmarking geschrieben. Diese misst die Ausführungszeit für eine C#-Methode. Dies findet mit einer bestimmten Anzahl an Wiederholungen statt, die als Eingabe festgelegt werden kann. Dieser Ansatz liefert jedoch nur begrenzt Aussagen über die Performance, da hier die Grafischen Ausgaben nicht mit berücksichtigt werden. Aus diesem Grund haben wir ein etwas ausgefeilteres Programm geschrieben, wo auch die grafischen Ausgaben berücksichtigt werden. Dabei verwenden wir ein weiteres Werkzeug: MonoDevelop, da der Test lediglich für Mono geschrieben wurde. Als Testsysteme standen ein Notebook und ein Desktop-PC bereit. Die Hardware ist bei beiden etwa 2 Jahre alt (2012).

ET_1 Erzeugen großer Knoten

Testhardware: Notebook ↗ Systemeigenschaften, Anhang auf S. ??

Unsere Benchmark-Klasse ohne Grafik-Ausgabe liefert beim Erzeugen quadratischer Knoten unterschiedlicher Größen:

(min: Kürzeste Zeit, max: Längste Zeit, avg: Mittlere zeit, out: Erster Ausreißer/Caching)

Knoten-Erzeugen: Knoten mit 100 Kanten, 100 WH:

max = 600210 NS >= 0 MS >= 0 S

min = 57915 NS >= 0 MS >= 0 S

avg = 78165 NS >= 0 MS >= 0 S

out = 50158035 NS >= 50 MS >= 0 S

Knoten-Erzeugen: Knoten mit 1000 Kanten, 1000 WH:

max = 2246940 NS >= 2 MS >= 0 S

min = 532575 NS >= 0 MS >= 0 S

avg = 607905 NS >= 0 MS >= 0 S

out = 56641680 NS >= 56 MS >= 0 S

Knoten-Erzeugen: Knoten mit 10000 Kanten, 10000 WH:

max = 19330650 NS >= 19 MS >= 0 S

min = 5473980 NS >= 5 MS >= 0 S

avg = 7617240 NS >= 7 MS >= 0 S

out = 54172395 NS >= 54 MS >= 0 S

Knoten-Laden: Knoten mit 100 Kanten, 100 WH:

max = 1639440 NS >= 1 MS >= 0 S

min = 666630 NS >= 0 MS >= 0 S

avg = 913275 NS >= 0 MS >= 0 S

out = 62651880 NS >= 62 MS >= 0 S

Knoten-Laden: Knoten mit 1000 Kanten, 100 WH:

max = 7217100 NS >= 7 MS >= 0 S

min = 4754295 NS >= 4 MS >= 0 S

avg = 5197365 NS >= 5 MS >= 0 S

out = 80073360 NS >= 80 MS >= 0 S

Knoten-Laden: Knoten mit 10000 Kanten, 100 WH:

max = 93382065 NS >= 93 MS >= 0 S

min = 64106235 NS >= 64 MS >= 0 S

avg = 69729660 NS >= 69 MS >= 0 S

out = 148111740 NS >= 148 MS >= 0 S

Unter Berücksichtigung der grafischen Ausgabe dauerte das Erzeugen von Knoten mit ...

... 500 Kanten: Zwischen 15 und 19 MS.

... 1000 Kanten: Zwischen 27 und 32 MS.

... 5000 Kanten: Zwischen 140 und 208 MS.

... 10000 Kanten: ∞ MS.

Testhardware: Desktop ↗ *Systemeigenschaften, Anhang auf S. ??*

Unter Berücksichtigung der grafischen Ausgabe dauerte das Erzeugen von Knoten mit ...

... 500 Kanten: Zwischen 5 und 10 MS.

... 1000 Kanten: Zwischen 12 und 18 MS.

... 5000 Kanten: Zwischen 100 und 160 MS.

... 10000 Kanten: Zwischen 255 und 301 MS.

Spielbarkeitstests

Kritiken und Kommentare von Testspielern.

Testperson 1

Schülerin ohne große Spielerfahrung oder Nähe zur Informatik. Das Spielkonzept wurde erst nach dem Lesen der Spielanleitung verständlich. Die grundlegende Steuerung wurde als nicht intuitiv empfunden. Wie Knoten transformiert werden wurde dagegen schnell entdeckt. Besondere Funktionen wie Selektion mehrerer Kanten oder das Einfärben von Kanten schaute sie im Einstellungsmenü nach. Auch das Aufrufen des Menüs über Escape war der Testerin intuitiv klar. Bei den einfacheren Challenges gab es keine Probleme. Bei den schwereren war etwas Einarbeitungszeit für die Orientierung im Raum notwendig. Nach ein paar Minuten waren schwerere Challenges auch nicht mehr problematisch. Während des Tests traten kleinere Fehler auf, die den Spielablauf jedoch nicht weiter störten und schnell behoben werden konnten. Erstens wurde die Drehungen im linken Bereich nicht sofort in den rechten übertragen. Drehungen im rechten Bereich hingegen wurden direkt im linken Bereich übernommen. Das zweite Problem war, dass bei mehreren Drehungen im linken Bereich die Rotation durch die verzögerte Darstellung im rechten Bereich etwas abwich.

2.5. Nicht getestet

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

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](#).

Komponententests

Von den Klassen, die wir für das Komponententesten in Betracht gezogen haben, mussten wir diejenigen ausschließen, die für einen entscheidenden Teil ihrer Funktionalität eine oder mehrere Instanzen der Klassen Game, GraphicsDeviceManager, GraphicsDevice oder ContentManager benötigen. Das bedeutet, dass sie Instanzen dieser Klassen entweder im Konstruktur erstellen, im Konstruktor 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 erbenden Klassen wegen der intensiven Nutzung von GraphicsDevice und GraphicsDeviceManager sowie teilweise Instanzen von Effect-Klassen, die den Zugriff auf Shader kapseln und ebenfalls von GraphicsDevice und ContentManager abhängen.

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, eventbasiert aufgerufenen Methoden, Listen von GameModels zu erstellen, sind damit von ContentManager und vom GraphicsDevice abhängig.

2.6. Statistik

2.6.1. Abdeckung

Auf den folgenden Seiten steht die Kurzversion des aus den Daten von ⁴ OpenCover generierten Berichts zur Testabdeckung durch Komponententests. Der prozentuale Anteil bezieht sich hierbei auf die Anzahl der abgedeckten Zeilen Code (LOC), aller als relevant eingestufter Komponenten. In Tabelle 2.2 sind die Ergebnisse aufgelistet.

Tabelle 2.2.: Testabdeckung durch Komponententests

Selektiv (lokal, .NET 4.5.1, XNA 4.0):	~ 86,20 %
Selektiv (online, Mono 3.2, MonoGame-SDL2):	~ 89,18 %
Gesamt:	~ 30,00 %

Die selektive Testabdeckung bezieht sich auf den Anteil aller Klassen die wir nicht herausgefiltert haben. Dabei verwendeten wir ein lokales- und ein Online-Werkzeug, die beide auf OpenCover basieren.

Online läuft die aktuellste Version von OpenCover unter Ubuntu 14.04 mit Mono 3.2, lokal läuft OpenCover unter .NET 4.5.1. Lokal erreichen wir eine Abdeckung von ~ 86,20 % und online ~ 89,18 %.

Der Unterschied kommt hauptsächlich durch die unterschiedlichen Compiler und Laufzeitumgebungen zustande, da Mono und .NET unterschiedlich effizient in sowohl der Kompilierung als auch in der Ausführung des Codes sind und jeweils auch im Debug-Modus unterschiedliche Anweisungen wegoptimieren.

Wir geben hier beide Werte an, da diese für uns auch laufend eine Selbstkontrolle darstellen. Die Gesamt-Testabdeckung wurde unter Mono ermittelt und ist um einiges niedriger als die selektiven Testabdeckungen. Dies liegt daran, dass wir alle GUI-Klassen selbst geschrieben haben (Widgets, etc.) und diese größtenteils nicht testen konnten. Wir verweisen an dieser Stelle nochmals auf die ausführliche Erklärung unter ⁴ Abschnitt 2.5, ab S. 23.

Summary

Generated on: 09.03.2014 - 12:33:55
Parser: OpenCoverParser
Assemblies: 2
Classes: 44
Files: 44
Coverage: 86.2%
Covered lines: 2097
Uncovered lines: 335
Coverable lines: 2432
Total lines: 7153

Assemblies

Knot3	89%
Knot3.Game.Audio.Knot3AudioManager	100%
Knot3.Game.Audio.Knot3Sound	100%
Knot3.Game.Data.Axis	100%
Knot3.Game.Data.Challenge	70%
Knot3.Game.Data.ChallengeFileIO	90.5%
Knot3.Game.Data.ChallengeMetaData	89%
Knot3.Game.Data.CircleEntry`1	92.3%
Knot3.Game.Data.CircleExtensions	100%
Knot3.Game.Data.Direction	100%
Knot3.Game.Data.Edge	100%
Knot3.Game.Data.Knot	84.8%
Knot3.Game.Data.KnotFileIO	85%
Knot3.Game.Data.KnotMetaData	86%
Knot3.Game.Data.KnotStringIO	86.2%
Knot3.Game.Data.Node	73.7%
Knot3.Game.Data.NodeMap	88.5%
Knot3.Game.Utilities.FileIndex	100%
Knot3.Game.Utilities.SavegameLoader`2	100%
Knot3.Framework	83%
Knot3.Framework.Audio.AudioManager	86.9%
Knot3.Framework.Audio.LoopPlaylist	69.2%
Knot3.Framework.Audio.OggVorbisFile	86%
Knot3.Framework.Audio.SilentAudioManager	0%
Knot3.Framework.Audio.Sound	100%
Knot3.Framework.Core.Camera	65.3%
Knot3.Framework.Core.DisplayLayer	98.1%
Knot3.Framework.Core.TypesafeEnum`1	100%
Knot3.Framework.Math.Angles3	100%
Knot3.Framework.Math.BoundingCylinder	90.4%
Knot3.Framework.Math.Bounds	94.5%
Knot3.Framework.Math.RayExtensions	68%
Knot3.Framework.Math.ScreenPoint	51.8%
Knot3.Framework.Platform.SystemInfo	100%
Knot3.Framework.Storage.BooleanOption	100%
Knot3.Framework.Storage.Config	66.6%
Knot3.Framework.Storage.ConfigFile	100%
Knot3.Framework.Storage.DistinctOption	100%
Knot3.Framework.Storage.FileUtility	97.2%
Knot3.Framework.Storage.FloatOption	92%
Knot3.Framework.Storage.IniFile	96.9%
Knot3.Framework.Storage.KeyOption	100%
Knot3.Framework.Storage.Language	87.8%
Knot3.Framework.Storage.LanguageOption	76.9%

Knot3.Framework.Storage.Localizer	87.5%
Knot3.Framework.Storage.Option	100%

3. Fehler

3.1. Übersicht

3.1.1. Klassifizierung

Programmfehler

Fehler im Programm.

Anzeigefehler

Fehlerhafte grafische Darstellung.

Fehlender Bestandteil

Dabei handelt es sich um Dinge, die in der Implementierungsphase nicht umgesetzt werden konnten und noch ausstehen. Bedürfnisse die z.B. wegen eines anderen Problems auftreten fallen auch in diese Kategorie.

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-Issues

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 (Reguläre Ausdrücke) .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. Gendarme wird unter Ubuntu 14.04 ausgeführt und arbeitet mit den mit Mono 3.2 kompilierten Binaries von Knot3.

3.3. Protokoll

Beschreibungen zu den Fehlern, gegliedert nach Fehlerklassen.

Programmfehler

#102

Unter dem Creative-„Save As“-Dialog sind Knotentransformationen u. weitere Eingaben möglich.

Lösung: Gamescreen ignoriert Eingaben bei geöffnetem Dialog. Dies wurde durch das Hinzufügen einer Boolean für modale Dialoge erreicht.

#95

Mehrere Pause-Menüs (auch übereinander) öffnen. Esc schließt nicht aktuellen Dialog, sondern öffnet neues Pause-Menü ↗ s. Abb. A.2, auf S. ??.

Lösung: Die Zeichenreihenfolge wurde angepasst, sodass der aktuelle Dialog Esc abfängt.

#93

Erstellen einer Challenge aus zwei gleichen Knoten

Lösung: Es wird vor dem Erstellen der Challenge auf Knotengleichheit geprüft, sodass keine ungültigen Challenges mehr erstellt werden können.

#97

Beim Laden eines Knotens, bei einem zweiten Klick auf den Knoten verschwindet die Knoteninfo.

Lösung: Knoteninfo wird nicht mehr pauschal gelöscht, sondern nur wenn sie sich ändert.

#117

Einflussbereich für Musikeinstellung.

Regler reagiert auf Klicks im gesamten Screen-Bereich.

Lösung: Widget prüft ob Maus Regler bewegt.

#83

Knoten nach Außerhalb des umgebenden Würfels bauen.
Kannten können aus der Skybox heraus gezogen werden.

Lösung: Die Skybox vergrößert sich nun automatisch.

#107

„Redo/Undo“ nach bestandener Challenge immer noch interaktiv.

Lösung: Button-Sichtbarkeit wird beim Beenden der Challenge auf „false“ gesetzt.

#84

„Überzoomen“, sehr nahes ranzoomen ist problematisch, flippt manchmal die Kamera.

Lösung: Zoomen begrenzt auf einen Minimalwert.

#103

Eingabe von Whitespace (z.B. Leerzeichen, eins oder mehrere) dort wo Strings vom Spieler festgelegt werden.

Lösung: Whitespace-only Eingaben werden nun nicht mehr erlaubt.

#105

Tastaturbelegung: Festlegen der gleichen Taste für mehrere Aktionen.

Lösung: Nun kann eine Taste nur einer Aktion zugewiesen werden.

#147

Spielbarkeit: Knotentransformationen, Übergänge, Kamera.

Lösung: Nun kann eine Taste nur einer Aktion zugewiesen werden.

Anzeigefehler

#82

Es ist immer noch möglich „fehlende Modelle im Knoten zu erzeugen“ ↗ s. Abb. A.6, auf S. ??.

Lösung: Skalierung der Modelle wurde sichergestellt.

#92

Fehlerhafte Darstellung bei Übergängen. Wenn sich die rohrartige Geometrie in einer Gitterkreuzung trifft, sind je nach Perspektive z.B. Überlappungen wahrnehmbar ↗ s. Abb. A.5, auf S. ??.

Partielle Lösung: Neue Modelle und neue Skalierungen wurden eingesetzt.

#96

„Start“-Schaltfläche ist inkonsistent zum restlichen Design

Lösung: Fehlende Linien wurden hinzugefügt.

#101

Redo-Button bei Challenge-Start.

Redo-Butten von Anfang an sichtbar.

Lösung: Sichtbarkeit zu Beginn auf „false“.

#119

Verschieben des Pause-Menüs.

Die Ränder des Pause-Menüs verschieben sich nicht ↗ s. Abb. A.3, auf S. ??.

Lösung: Ränder werden dynamisch positioniert.

#108

Challenges: Highscores und Menüeinträge nicht getrennt voneinander/wahrnehmbar.

Lösung: Menüeinträge sind nun am unteren Rand des Dialogs.

Fehlende Bestandteile

#78

„Exit“-Symbol fehlt.

Lösung: Neues Symbol erstellt und zum Start-Screen hinzugefügt.

#141

Textbox für auto-umgebrochenen Fließtext.

Lösung: Textbox wurde implementiert und hinzugefügt.

3.4. Statistik

⌘ Gendarme

Anzahl der durch Gendarme gefundenen Probleme verteilt über die Monate von Januar bis März 2014:

	Januar	Februar	März
	> 1200	~ 600	~ 300

⌘ „Issues“

Es wurden insgesamt 162 Tickets bei den GitHub-„Issues“ erstellt. Bis zum Ende der QS-Phase wurden 137 Tickets erfolgreich bearbeitet und geschlossen. Nur 25 Tickets blieben offen.

Anzahl geschlossener Tickets je Auszeichnung:

	Bug	Missing	Design
	53	5	3

Anzahl noch offener Tickets je Auszeichnung:

	Bug	Missing	Design
	5	3	3

4. Änderungen

4.1. Protokoll

4.1.1. Geändert

Fehler die wir verbessert haben oder Ergänzungen werden hier beschrieben. Kleinigkeiten fließen nicht in das Protokoll ein.

Transformations-Vorschau

Problem: Während der Spieler eine Transformation durchführt, konnte er nicht erkennen, wie das Ergebnis dieser Transformation aussehen wird. Dies wurde im Pflichtenheft durch das Anzeigen einer Schattenvorschau jedoch gefordert.

Änderung: Wir zeigen nun Änderungen direkt an so wie sie übernommen werden würden, wenn man den Zug beendet. Dies erforderte einen erheblichen Umbau in der Art wie ein Zug ausgeführt wird. Während vorher separat geprüft wurde ob eine Transformation gültig ist und die eigentliche Transformation erst beim beenden des Zuges durchgeführt wurde war dieses System für eine Vorschau ungeeignet. Jetzt wird versucht die Transformation durchzuführen und sowohl der Erfolg bzw. Misserfolg sowie ein neuer veränderter Knoten zurückgegeben, der als Vorschau verwendet werden kann und nach Abschluss des Zuges den aktuellen Knoten ersetzt. Abschließend haben wir uns dagegen entschieden die nicht geänderten Teile des alten Knotens anzusehen, da es sowohl erheblich mehr Rechenleistung, deren Bedarf ohnehin schon durch die direkte Vorschau erheblich gestiegen ist, als auch der Übersichtlichkeit schadet. Die zusätzlichen Kannten, selbst halb-transparent, hätten das Blickfeld zusätzlich behindert ohne dabei einen Nutzen zu haben.

OpenAL und SDL2

Auf Systemen, auf denen kein OpenAL bzw. SDL2 vorhanden ist, wird dieses automatisch heruntergeladen und ins Installationsverzeichnis von Knot3 entpackt. Dann ist Knot3 nach einem Programmneustart spielbar.

Knot3 nutzt nun auch eine neuere Version von MonoGame-SDL2, die die freie OpenAL-Implementierung OpenAL Soft und nicht die von Loki Software / Creative Labs nutzt. Siehe:

<https://github.com/flibitijibibo/SDL2-CS/pull/54>

Die von uns zur Kompilierung verwendeten Versionen der Abhängigkeiten sind SDL Version 2.0.2, SDL2_image Version 2.0.0 und OpenAL Soft Version 1.15.1.

Spielbeschreibung

Problem: Es gibt keinen Tutorial-Mode, indem dem Spieler die grundlegenden Spielmechaniken erklärt werden.

Änderung: Es gibt nun eine separate PDF („Spielanleitung.pdf“), welche die grundlegenden Spielmechaniken erklärt.

Lokalisierung

Problem: Es gibt keine Lokalisierung, alle Texte im Spiel sind immer auf Englisch.

Änderung: Es gibt nun im Content-Verzeichnis für jede unterstützte Sprache je eine ini-Datei, die eine Zuordnung zwischen den englischen Strings und den in die jeweilige Sprache übersetzten Strings enthält und vom Spiel eingelesen wird.

Tastenabfrage

Problem: Wir haben festgestellt, dass unser gewähltes Abtast-Intervall von 100 Millisekunden zu kurz ist für normale Tasteneingaben. Bei normalen Tippen konnte es passieren das nun die Taste zweimal abgetastet wurde. Was eingaben von Spie lernamen erschwerte.

Änderung: Das Intervall wurde auf 200 Millisekunden erhöht um dieses Problem zu verhindern.

Language

Problem: Ein erstelltes Language-Objekt konnte seine Attribute nicht mehr ändern, nachdem es erstellt wurde.

Änderung: Die Attribute verweisen nur auf die entsprechende Datei in der alle Attribute in der aktuellsten Fassung stehen. Somit werden die Attribute nun direkt aus der Datei ausgelesen wenn sie benötigt werden.

Farbcodierung im Level-Dateiformat

Problem: Es konnten keine Farben in RGB angegeben werden. Es wurden immer RGBA werte erwartet. Falls es nur 6 Hexadezimal-Zahlen (RGB) waren wurde eine Exception ausgelöst.

Änderung: Es wird nun überprüft ob es sich um RGB oder RGBA handelt.

Große IF-ELSE-Blöcke

Problem: In der Dekodierung und Kodierung der Richtung beim Laden bzw. Speichern von Knoten in das Dateiformat wurde die Zuordnung jeweils durch einen großen IF-ELSE-Block erledigt.

Änderung: Es gibt jetzt ein Dictionary mit den Zuordnungen welches für beide Funktionen Verwendung findet. Das objektorientierte Programmier-Paradigma wird so besser umgesetzt.

Installation/Deinstallation

Problem: Im Pflichtenheft war eine automatische Installation/Deinstallation und Testfällen dazu vorgesehen.

Änderung: Während der Implementierungsphase wurde dieses Feature nicht implementiert. Wir haben uns darauf geeinigt das Spiel als ein Archiv auszuliefern. Es gibt keine automatische Installation/Deinstallation. Die Installation erfolgt vom Nutzer durch Entpacken des Archivs.

Die Abhängigkeiten von Knot3 werden teilweise mitgeliefert (die Programmbibliotheken von MonoGame-SDL2, OggSharp und Ionic.Zip) und teilweise automatisch beim ersten Start des Spiels heruntergeladen und installiert (OpenAL Soft und SDL2) - Siehe „OpenAL und SDL“.

Die Deinstallation nimmt der Spieler manuell auf seinem System vor. Dazu muss zunächst das Programmverzeichnis gelöscht werden. Wenn auch die Konfigurations-

dateien gelöscht werden sollen, so kann dazu das in der MonoGame-Version mitgelieferte ConfigReset-Tool zu diesem Zweck genutzt werden.

Error-Dialog

Problem: Das Spiel stürzte bei unvorhergesehenen Fehlern ab.

Änderung: Wir zeigen nun die Fehler die auftreten in einem extra eingefügten Error-Dialog an. Dieser Error-Dialog zeigt die von der geworfenen Exception übergebene Nachricht an. Das Spiel stürzt nicht mehr ohne Vorwarnung ab.

Programm-Icon

Änderung: Die ausführbare (.exe) Datei hat ein Icon erhalten.

Zurücksetzen der Einstellungen

Änderung: Falls das Spiel nicht mehr startet, kann die mitgelieferte ConfigReset.exe zum Zurücksetzen der Einstellungen genutzt werden. Diese ist nur in der MonoGame-Version enthalten.

4.1.2. Nicht verschönert

Während der QS-Phase haben wir nach dem Motto „Juice It Up“ Möglichkeiten geprüft, das Knot3-Spiel zu verschönern. Da die Zeit jedoch größtenteils zur Fehlerkorrektur genutzt wurde, konnten diese nicht mehr umgesetzt werden. Dennoch möchten wir die genannten Vorschläge hier dokumentieren.

Blinkende Sterne

Die Umgebung im Creative- oder im Challenge-Mode ist recht neutral. Wir haben eine spaciege an den Weltraum oder einen Sternenhimmel erinnernde Skybox. Außer die vom Spieler initiierten Knotentransformationen gibt es keine grafischen Änderungen. Als einfache Erweiterung wurde angedacht, einige der bereits vorhandenen Sterne im Hintergrund zum Blinken zu animieren. Der Vorteil unserer Implementierung ist allerdings, dass der Spieler durch nichts abgelenkt wird.

Menüführung und Menü-Stil

Im Hauptmenü, ↗ s. Anhang, S. 44, wollten wir durch eine Knotenschaltfläche noch die Möglichkeit bieten, sich die Mitwirkenden anzeigen zu lassen. Diese Änderungen ist zu aufwendig, da die Erstellung der grafischen Oberflächen bei unserer Implementierung mit vielen hart-kodierten Werten viel Zeit beansprucht. Das Einfügen eines Credits-Buttons verschöbe die bereits vorhandenen Komponenten und erfordert so einen Neuentwurf des ganzen Hauptmenüs.

Mit dieser Überlegung haben wir gleichzeitig geprüft, wie hoch der Aufwand für eine optisch ansprechendere Darstellung der Menüs wäre. Wir haben dazu ein Mock-Up für das Hauptmenü erstellt, ↗ s. Anhang, S. 44. Wie bereits erwähnt ist dafür viel Handarbeit nötig, was das Zeit-Budget nicht mehr zulässt.

5. Ausnahmen

Selbst für von uns unkontrollierbare oder unerwartete Probleme haben wir eine Vorsichtsmaßnahme getroffen. Sollten während des Spiels derartige Exceptions auftreten werden diese abgefangen und eine Meldung auf dem Bildschirm angezeigt.

5.1. Behandlung

Probleme von denen wir Bescheid wissen werden so gehandhabt, dass die ungültige Aktion vom Spieler nicht durchführbar ist. Auch von uns unkontrollierbare Situationen, wie z.B.

- Betriebssystem stellt keinen Dateispeicher mehr zur Verfügung.
- Speicherstand wird während des Spiels gelöscht.

oder unerwartete Fehler führen nicht direkt zu einem Absturz. In solchen Fällen greift ein „Catch-All“-Mechanismus und der Spieler wird über das Problem informiert. Dabei entsteht der psychologische Vorteil, dass Fehler deren Ursache nicht im Spiel liegt als solche gemeldet werden und nicht Knot3 anzulasten sind.

5.2. Meldungen

Für unkontrollierbare oder unerwartete Fehler werden allgemeine Meldungen durch einen ErrorScreen angezeigt. Diese Nachrichten enthalten die Beschreibung, welche die auftretende C#-Exception mit sich liefert.

6. Schluss

6.1. Bewertung

Die Qualität des Spiels Knot3 ist verbessert worden. Durch eine Komponententestabdeckung von über 80 % und Funktionstests für alle elementaren Spielfunktionen ist jetzt die einwandfreie Funktion von Knot3 gewährleistet.

Alle Grundfunktionen sind getestet und funktionieren zuverlässig. Bei der Abnahme in Spielbarkeitstests durch menschliche Testspieler hat sich gezeigt, dass unser Produkt, wenn auch nicht perfekt, für eine interessante Spielerfahrung geeignet und für den Einsatz bereit ist.

+In den Negativtests sichern wir das Programm weiter ab. Das Spiel ist so robust gegen die wahrscheinlichsten Störfälle. Die Algorithmen für die Knotentransformationen erfüllen ihre Aufgaben und es ist nicht möglich ungültige Knoten zu erzeugen (NT_10, NT_30, Komponententests). Die Robustheit wird zudem in Extremtests bei der Verarbeitung großer Datenmengen sichergestellt, so dass es bei realitätsnahen Knotengrößen mit um die 1000 Kanten zu keinen Problemen kommt (ET_1). Erst bei 5000 Kanten verringert sich die Spielgeschwindigkeit gravierend. Die Hardware der Testsysteme ist schon etwas älter. D.h. Knot3 läuft auf allen gängigen Systemen. Des weiteren ist durch Tests überprüft worden, dass bei nicht zugewiesenen Eingaben kein Fehlverhalten auftritt (NT_50). Der Spieler wird während dem gesamten Spiel dadurch unterstützt, dass wir ungültige Aktionen nicht zulassen.

Selbst bei Systemfehlern oder unerwarteten Fehlern geben wir einen Hinweis in Form einer Textmeldung an den Spieler. Diese Nachrichten sind im Nachhinein verwendbar, falls wir trotz allem etwas übersehen haben sollten, um den Fehler schnell zu finden und eine Aktualisierung bereit zu stellen.

+Ein weiterer Grund, der nicht nur für die Funktionstüchtigkeit spricht ist der Vorteil durch die Verwendung von automatischen Werkzeugen bei der Fehlersuche. Gendarme hat mit zu einer besseren Code-Qualität beigetragen und insgesamt auf mehr als 1000 Probleme hingewiesen. 75 % davon haben wir korrigiert. Darunter befanden sich auch zahlreiche kleinere Verbesserungen der Performance. Die restlichen 25 % sind nicht weiter schlimm. Denn Gendarme zeigt neben ernstzunehmenden Proble-

men oft kleinere syntaktische Feinheiten an. Wir nutzten diese Zeit besser für die Lösung der Probleme die wir bei unserer eigenen Fehlersuche gefunden haben. Dabei sind wir sehr präzise vorgegangen, unterstützt durch das Issuesystem von GitHub, Screencapturing, das parallele Schreiben der Komponententests und Berichten von Spieldetestern, wobei wir selbst häufig zum Testen gespielt haben. Zudem haben wir Testknoten zu grundlegenden Knotenarchitekturen erstellt und deren Bau überprüft. In einem Spiel kommen genau diese Situationen öfters vor. Wir erhoffen uns durch die einmalige Überprüfung, dass dies auch in komplexeren Fällen durchführbar ist.

Ein weiteres Kriterium für die hohe Qualität von Knot3 ist die hohe Portabilität des Spiels. Durch die Verwendung von MonoGame-SDL2 statt XNA lässt sich Knot3 auf alle Plattformen portieren, die von SDL2 unterstützt werden, im wesentlichen sind das die drei wichtigen Desktop-Plattformen (Windows, Linux, Mac OS X).

Um den Anteil an plattformspezifischem Code zu minimieren, werden so weit es geht externe Programmzbibliotheken genutzt und Abstraktionsklassen verwendet. Die Qualität des Codes wird auch dadurch deutlich, dass der Code des Spiels nicht nur mit MonoGame-SDL2 kompatibel sind, das auf aktueller freier Software aufbaut, sondern ein Großteil der Funktionalität auch noch mit der nicht mehr weiterentwickelten Implementierung der XNA-API von Microsoft verwendet werden kann.

Knot3 wurde von Anfang an parallel zur Entwicklung mit XNA/Visual Studio mit MonoGame und MonoDevelop/Xamarin Studio entwickelt. Das Entwickeln und Testen mit beiden Implementierungen der XNA-API hat uns die Stärken und Schwächen beider Implementierungen vor Augen geführt und wesentlich dazu beigetragen, dass wir Bugs in unserem Code gefunden haben, die wir bei der Entwicklung nur für eine Implementierung oder nur für Plattform übersehen hätten.

Besonders hilfreich war auch die Möglichkeit, sich die interne Implementierung diverser Klassen und Methoden in dem quelloffenen MonoGame-SDL2 ansehen zu können, da die Dokumentation von XNA 4.0 von Microsoft gerade bei komplizierten Themen wie der Nutzung von Shadern, Hardware Instancing relativ schlecht und spärlich ist. Auch haben sich die Autoren von MonoGame-SDL2 als sehr hilfreich erwiesen, wenn wir aufs Bugs oder Unterschiede zwischen MonoGame und XNA gestoßen sind. Gleichzeitig hat uns dies auch die Grenzen von XNA 4.0 vor Augen geführt, da in einer offiziell eingestellten proprietären Library keine Bugs mehr korrigiert werden konnten und ein tieferes Verständnis von den internen Abläufen in XNA notwendig war, ob diese Bugs umgehen zu können. Dies hat dann auch zu einer ausgereifteren Verwendung der XNA-API geführt, als wenn wir nur der offiziellen Dokumentation gefolgt wären, die diverse Probleme (etwa mit dem Anti-Aliasing oder der Shader-Unterstützung) nicht erwähnt.

A. Anhang

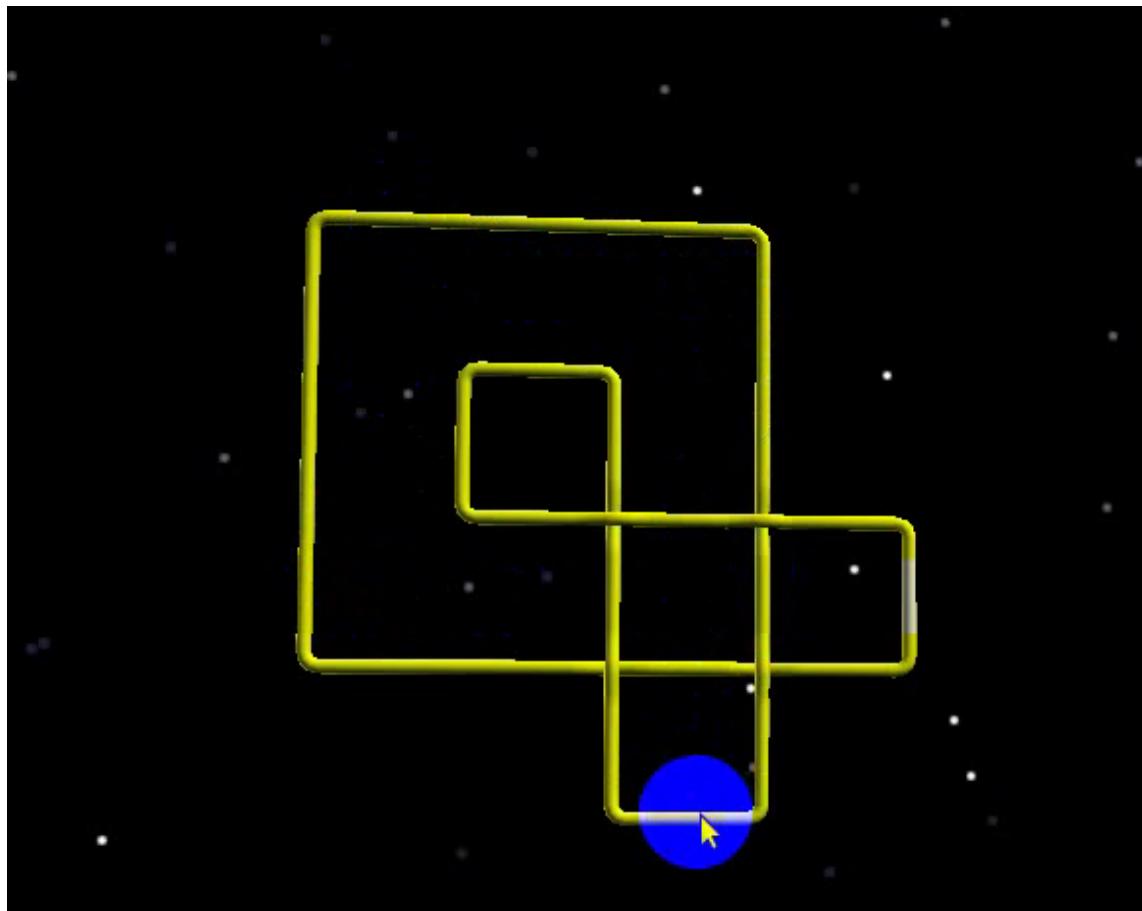
A.1. Aufnahmen

Eine kleine Auswahl der Bilder die wir beim Screencapturing erstellt haben.

A.1.1. Testknoten

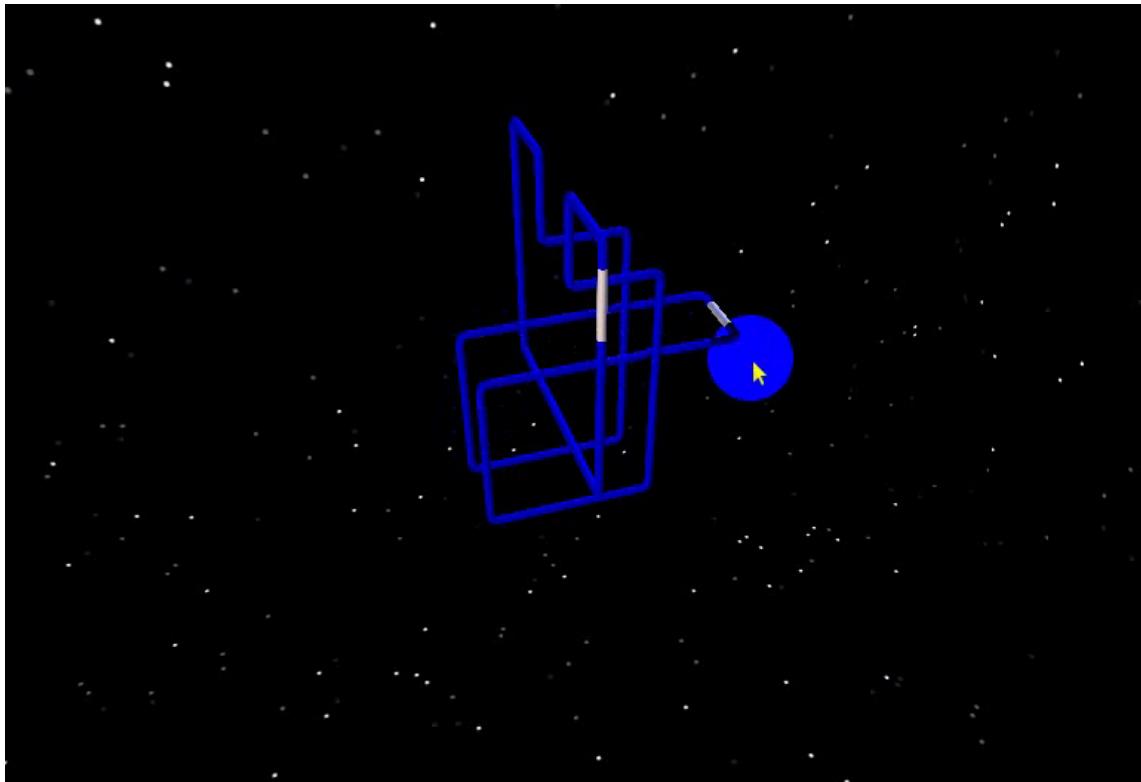
Ein Bilderkatalog der die Knoten zeigt, deren Erstellbarkeit wir explizit prüfen.
Hinweis: Mit Adobe Reader ab Version 9 ist es möglich den Knoten-Bau im PDF-Dokument als Animation abzuspielen.

„Überleger“



↗ Zu Funktionstest FT_30 (1.)

„Schlaufe“



↗ Zu Funktionstest FT_30 (2.)

Änderungen

Nicht geändert

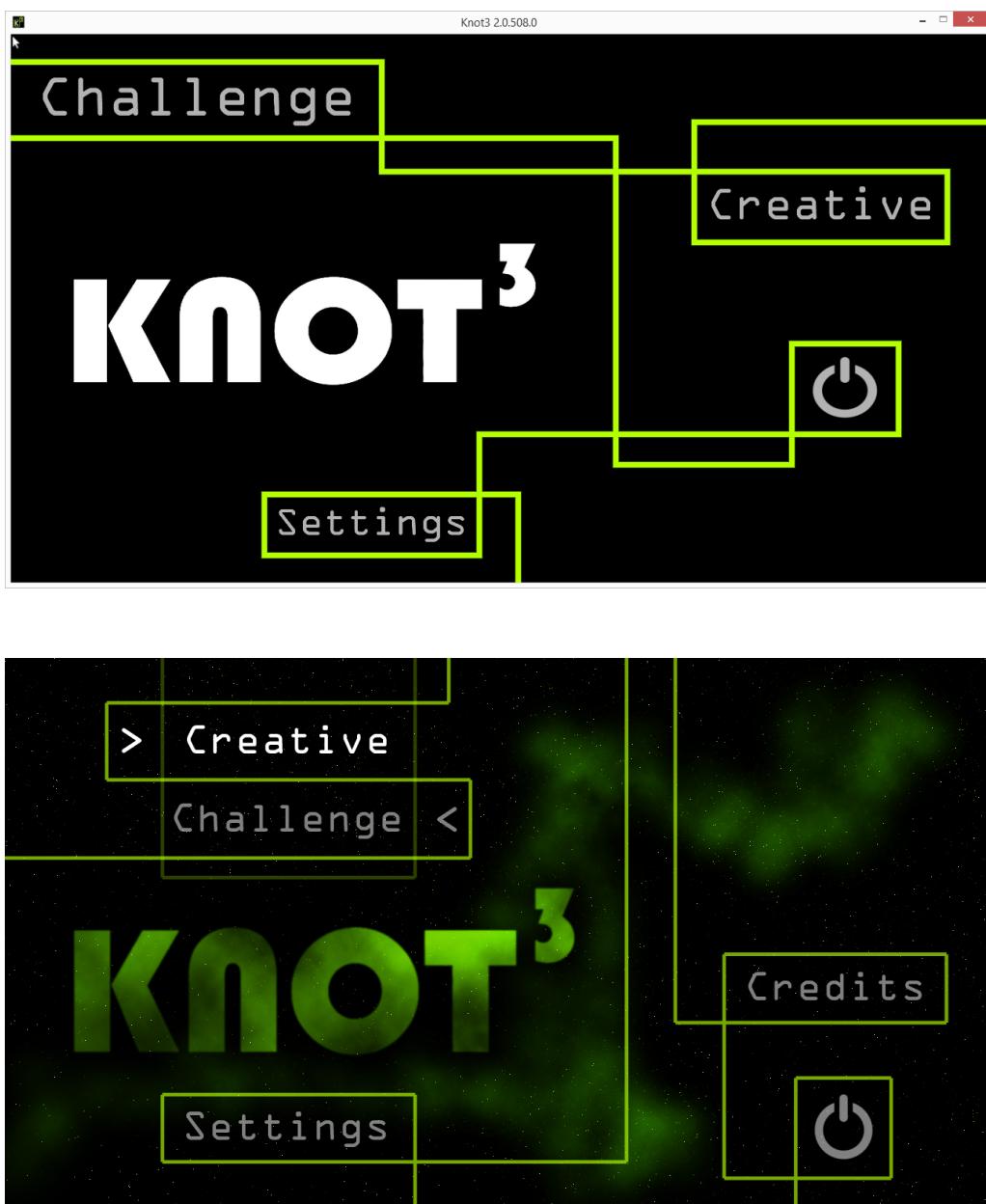


Abbildung A.1.: Unser Hauptmenü (oben).

Probeentwurf eines neuen Hauptmenüs (unten).

¶ Zurück zur Beschreibung unter Abschnitt 4.1.3, ab S. 38.

Anzeigefehler

Bilder von Fehlern, die wir durch grafisches Fehlverhalten entdeckten.

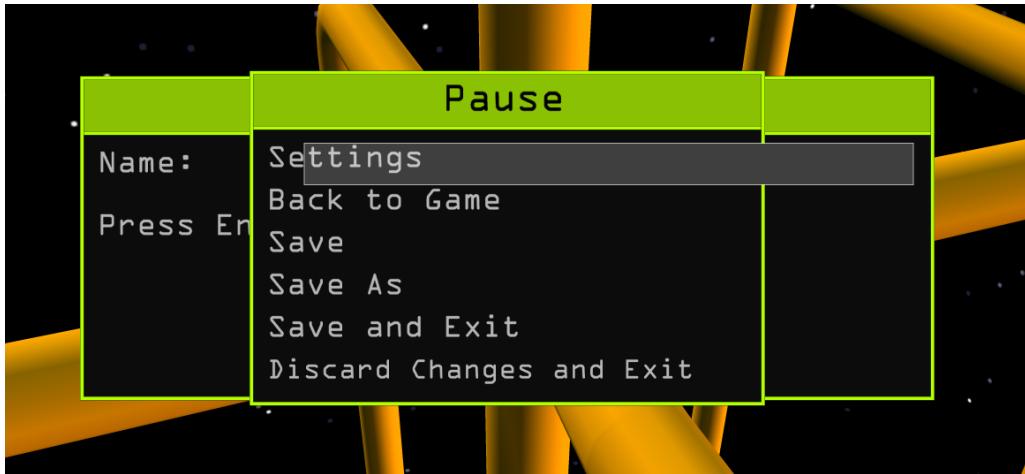


Abbildung A.2.: Öffnen eines Pausemenüs während eines Dialogs.



Abbildung A.3.: Fehlverhalten beim Verschieben eines Fensters.

¶ Zurück zum Fehlerprotokoll / Anzeigefehler unter Abschnitt ??, ab S. ??.

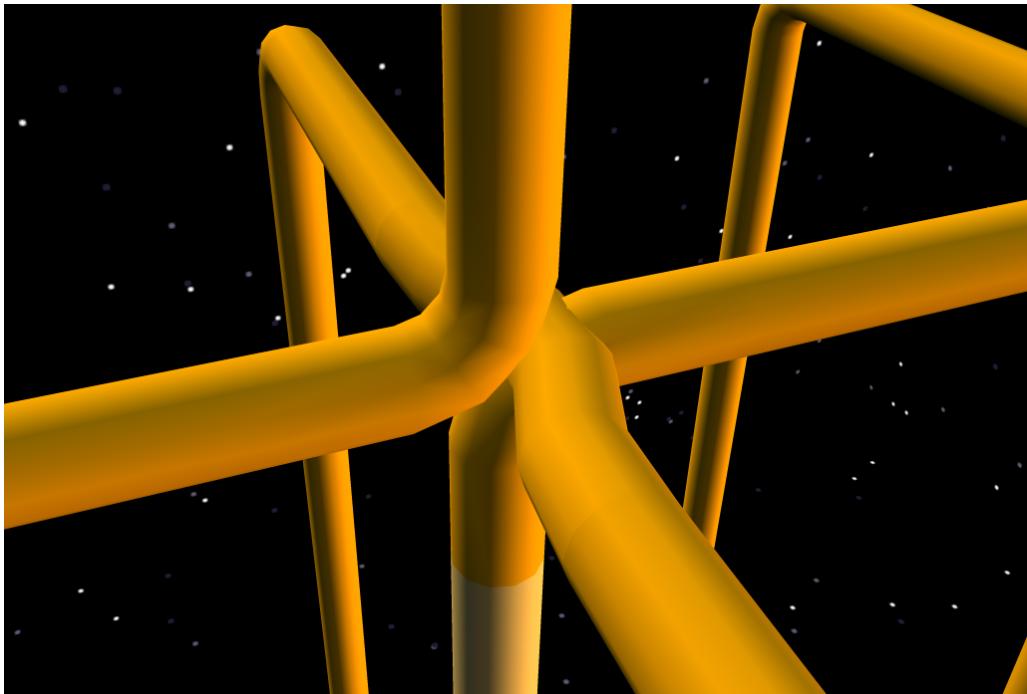


Abbildung A.4.: Falsches Modell beim Übergang.

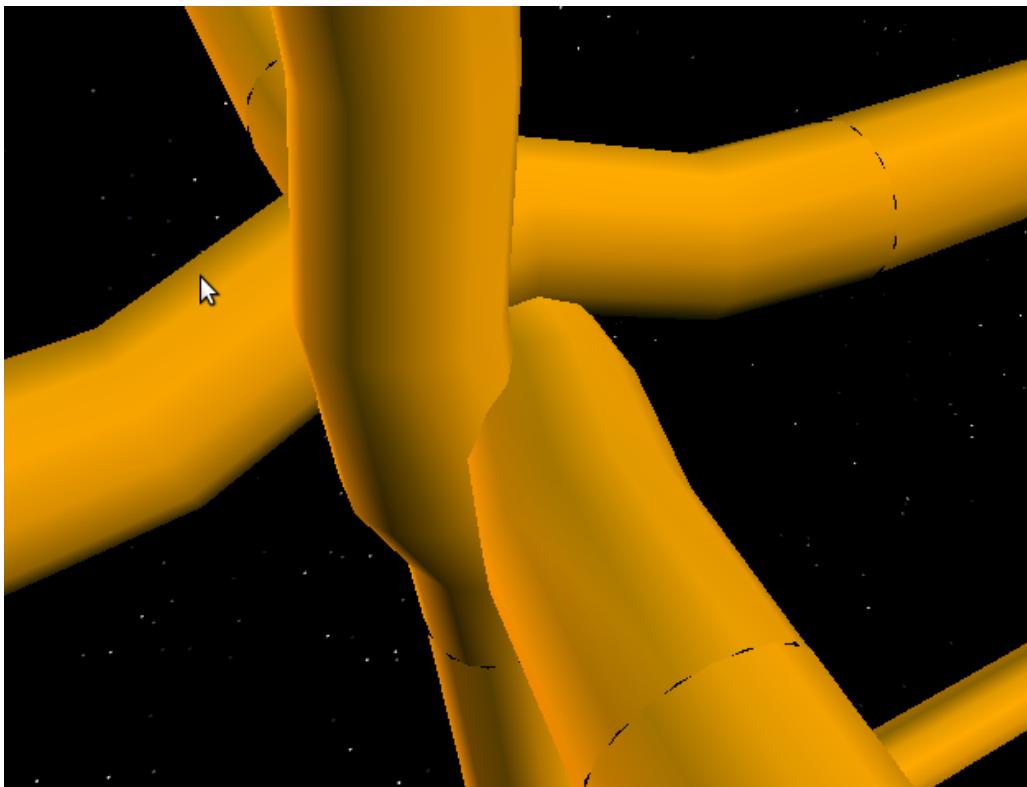


Abbildung A.5.: Verschmolzene Geometrie an den Übergängen.

↶ Zurück zum Fehlerprotokoll / Anzeigefehler unter Abschnitt ??, ab S. ??.

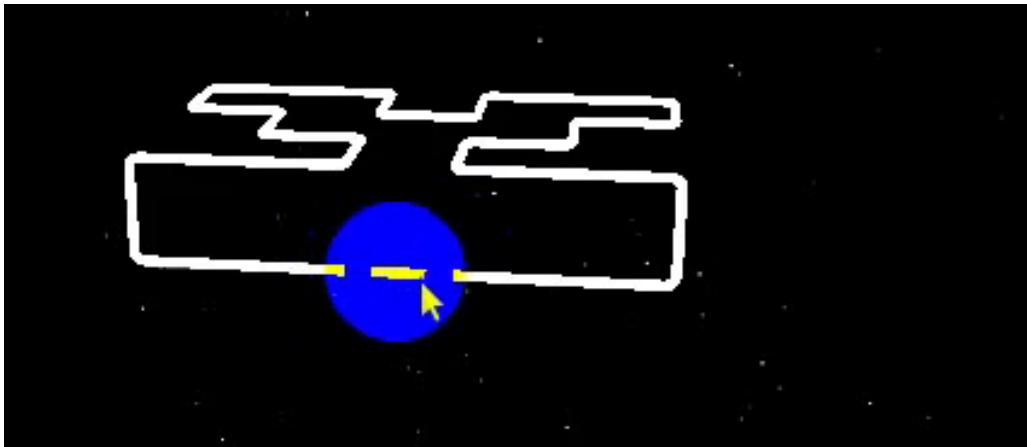


Abbildung A.6.: Löcher statt Übergänge.

↗ Zurück zum Fehlerprotokoll / Anzeigefehler unter Abschnitt ??, ab S. ??.

Testsysteme

Die Hardware, welche wir beim Testen in den Extremtests verwendet haben.

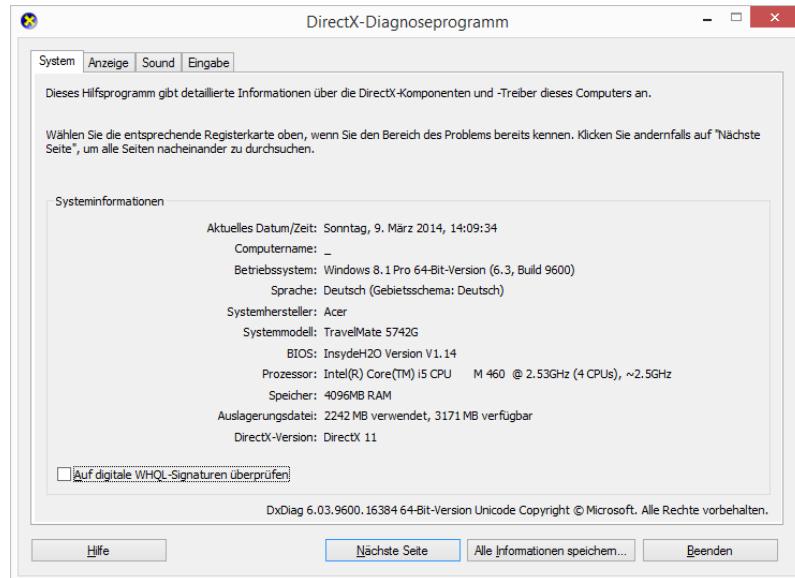


Abbildung A.7.: Systemeigenschaften - CPU (Notebook).

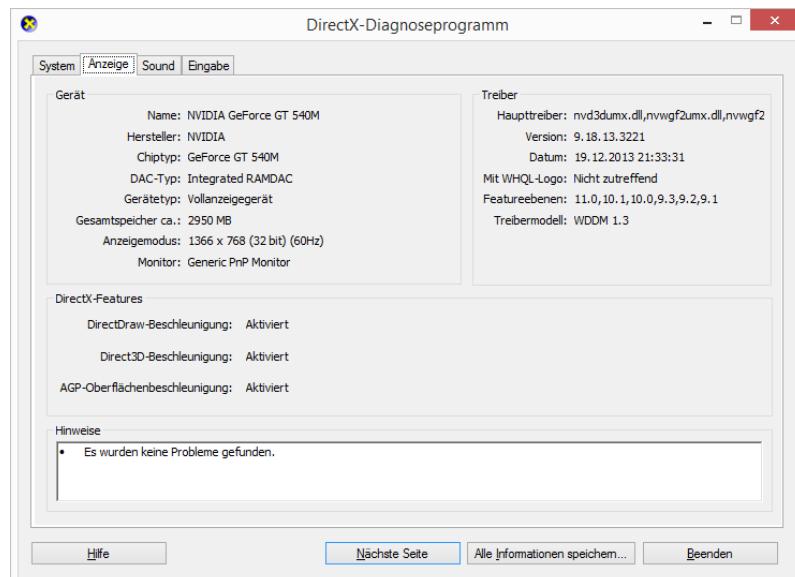


Abbildung A.8.: Systemeigenschaften - GPU (Notebook).

¶ Zurück zu den Extremtests unter Abschnitt 2.4, ab S. ??.

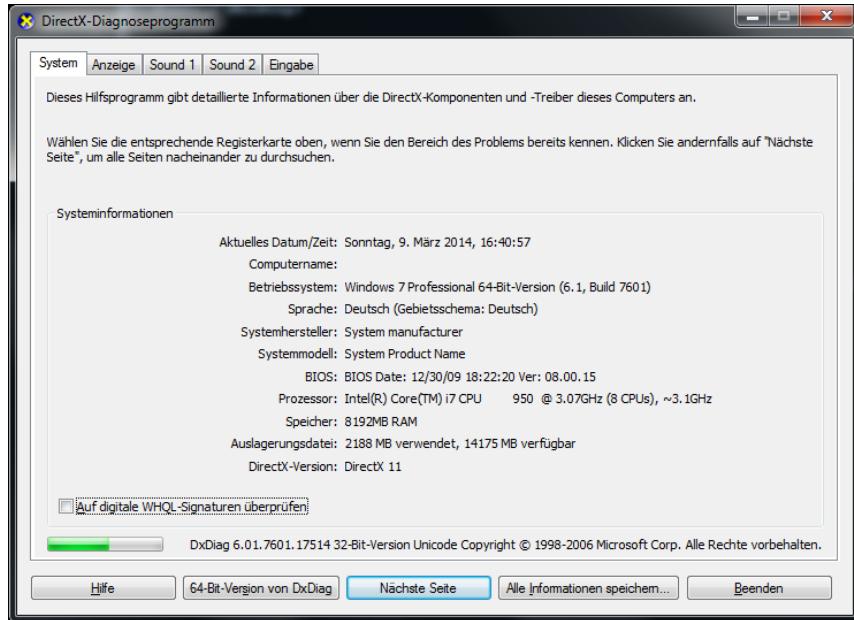


Abbildung A.9.: Systemeigenschaften - CPU (Desktop).

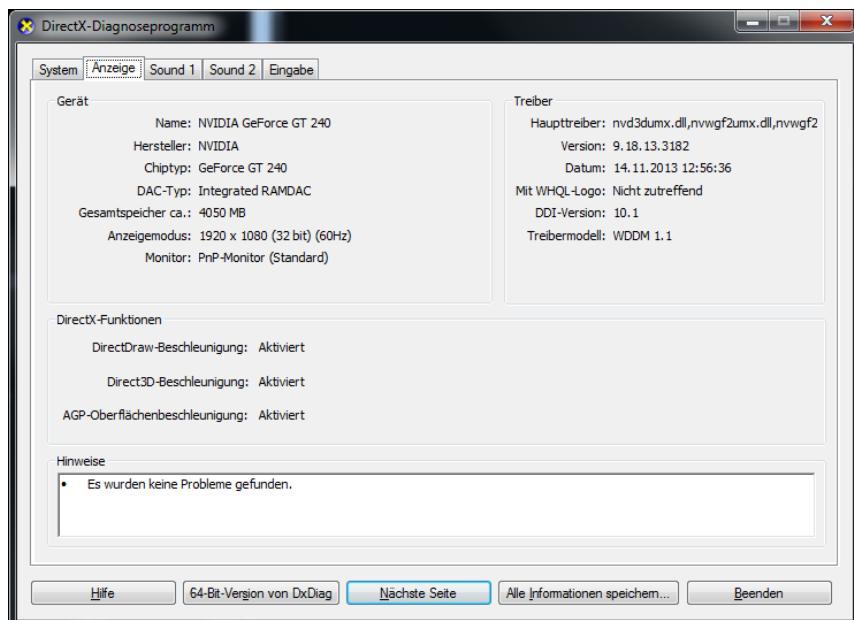
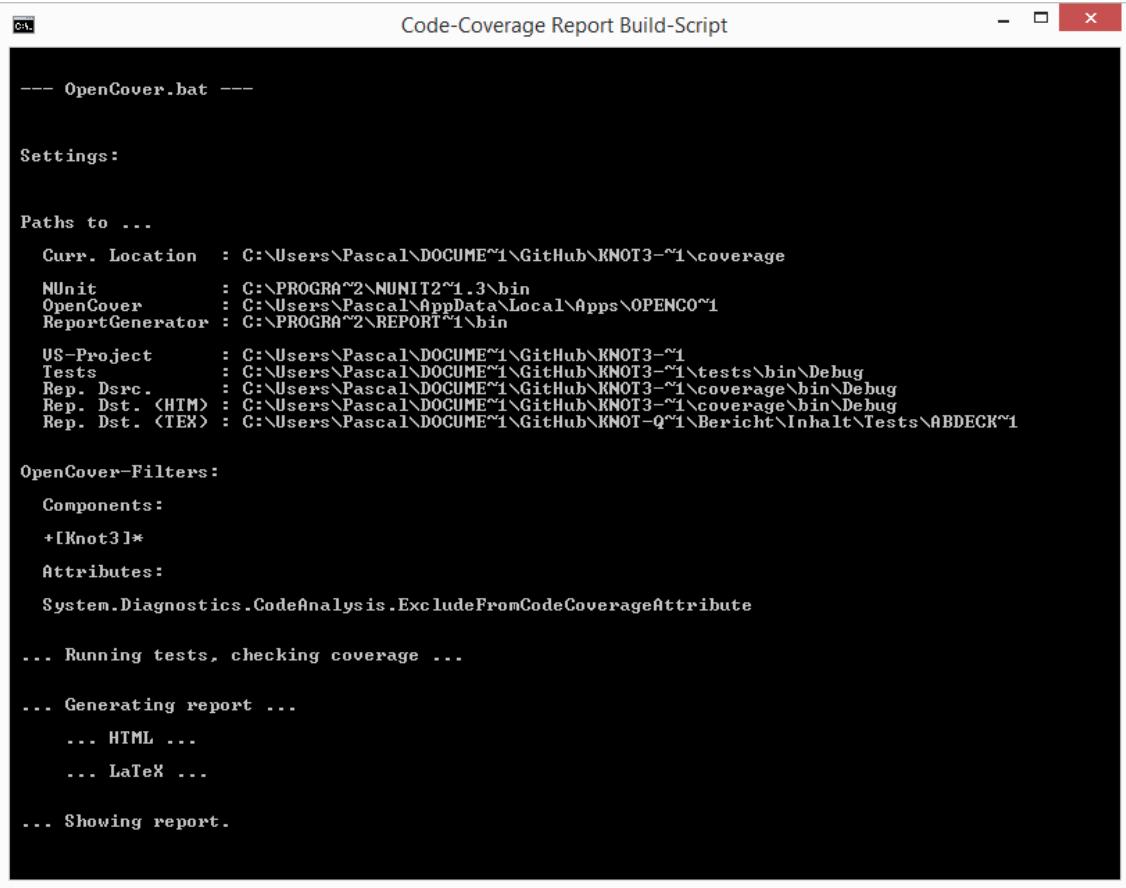


Abbildung A.10.: Systemeigenschaften - GPU (Desktop).

↶ Zurück zu den Extremtests unter Abschnitt 2.4, ab S. ??.

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\OPENCO^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-Q^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.11.: Build-Batch für die Erstellung des Testabdeckungs-Berichts.

↗ Zurück zur Beschreibung der Testwerkzeuge ab S. 6.