Implementierungsbericht
(V. 1.0)

KNOT^3
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

2. Februar 2014

Inhaltsverzeichnis

1 Einleitung

Nach Pflichtenheft und Entwurf stand nun die Implementierung auf dem Plan. Auf den folgenden Seiten beschreiben wir, wie das vonstatten ging, welche Änderungen wir vornehmen mussten und auf welche Probleme wir gestoßen sind. Durch die vielfältigen Möglichkeiten, die ein frei veränderbarer Knoten bietet, entstand eine hoch komplexe Interaktionsvielfalt, von der manche Bereiche im Entwurf nicht ausreichend Beachtung fanden, bzw. finden konnten. Dadurch sind einige Änderungen zum ursprünglichen Entwurf notwendig geworden, die sich aber vor allem auf Erweiterungen beziehen. Dennoch hat sich unser Entwurf als gut erwiesen, da alle Änderungen ohne Umbau der grundlegenden Strukturen integriert werden konnten.

2 Bericht

Während den Ferien und zu Beginn der Implementierung wurde von Tobias alles, was er schon wärend der anderen Phasen für den Prototypen entwickelt hat auf unseren Entwurf angepasst und implementiert.
Dadurch hatten wir früh eine umfangreiche Basis auf die wir aufbauen konnten, dies umfasste die grundlegende Klassenstruktur, die grundlegende Datenstruktur, die meisten Screens und Widgets, die 3D-Komponenten und -Objekte und den Standard-Rendereffekt.

Die Gruppenarbeit begann dann Mitte Januar. Als nächstes wurde die Datenstruktur vervollständigt und noch fehlende Widgets wie z.B. die SliderItems hinzugefügt. Dabei stießen wir bei der Datenstruktur auf keine unerwarteten Probleme, nur auf die Erwarteten. Das waren die Gültigkeitsprüfung eines Zuges und der Vergleich zweier Knoten.

Bei der Gültigkeitsprüfung des Zuges ergab sich, das es die beste Variante ist den Zug intern ohne Änderung an der Datenstruktur auszuführen und die entstandene Strktur auf Gültigkeit zu prüfen.
Beim Vergleich zweier Knoten hat der direkte Zugriff auf die Datenstruktur und die Art der Struktur den erwünschten erleichternden Effekt gehabt. Da jedes Element des doppelt verketteten Kreises als Startpunkt dienen konnte, wurde zum Vergleich einfach ein neuer, eindeutiger Startpunkt gewählt.

Bei den Widgets hingegen sind einige Varianten aufgetaucht die wir im Entwurf noch nicht bedacht hatten. Zum Beispiel gab es keine gute Variante reinen Text anzuzeigen, der nicht auf Eingaben regaieren sollte.

Außerdem mussten wir feststellen, dass wir für das Pausenmenü im Creative und im Challenge Mode doch zwei unterschiedliche Widgets brauchen, da sie sich intern stärker unterscheiden als erwartet. Dazu kamen noch verschieden Helfer, die einem die Arbeit nur sehr erleichtert haben wie Border, das die Umrahmung von Widgets einfach ermöglicht. Während dieser Zeit haben sich andere aus unserem Team mit der Auswahl der Musik beschäftigt. Der schnelle Fortschritt, den wir durch Tobias Arbeiten in den anderen Phasen und in den Ferien hatten, führte in der Mitte der Implementierungsphase zu einer stark erhöhten Bereitschaft viel der Arbeitszeit mit konzeptionellen Ideen zur Implementierung und mit Recherchen zu Musik, Grafiken und 3D-Modellen zu verbringen. Dies änderte sich erst als uns die Zeit längst eingehohlt hatte. Bei der Audio-API sind wir auf das Problem gestoßen, dass XNA nur den proprietären WMA-Codec oder reine WAV-Dateien unterstützt. MonoGame unterstützt allerdings auf Plattformen wie Linux aus lizenzrechtlichen Gründen keine WMA-Dateien, sondern nur WAV-Dateien. Da wir nicht vollständig darauf verzichten wollten, komprimierte Audiodateien verwenden zu können, haben wir eine modifizierte Version der Library OggSharp [1] eingebunden, um den freien Ogg-Vorbis-Codec verwenden zu können, der sich auch allein aus lizenzrechtlichen Gründen besser eignet als WMA. Nun wurden auch endlich Langzeitbaustellen angegangen, wie zum Beispiel die Kantenübergänge. Dies stellte sich als sehr aufwendig heraus, da keine Möglichkeit gefunden wurde die Ausrichtung der Übergänge Algorythmisch zu ermitteln, sondern jeder mögliche Übergang einzeln von Hand bearbeitet werden mussten.

[1] https://github.com/tobiasschulz/oggsharp

3 Hinzugefügte Klassen

Dieses Kapitel enthält eine Auflistung aller Klassen, die während der Implementierungsphase hinzugefügt wurden.

3.1 Core:

3.1.1 FloatOptionInfo

Grund: Analog zu BooleanOptionInfo brauchten wir auch eine Möglichkeit, Gleitkommazahlen in der Einstellungsdatei abzuspeichern.

3.1.2 KeyOptionInfo

Grund: Analog zu FloatOptionInfo brauchten wir für die Konfigurierbarkeit der Tastenbelegungen auch eine Möglichkeit, Tasten in der Einstellungsdatei abzuspeichern, repräsentiert durch das "KeysEnum von XNA.

3.1.3 IGameScreen

Grund: Um ein Mock-Objekt für GameScreen erstellen zu können, werden alle von außen verwendeten Eigenschaften und Methoden von GameScreen in einem Interface zusammengefasst und überall im Code wird IGameScreen verwendet. Dies erleichtert die Unit-Tests.

3.1.4 IMouseClickEventListener, IMouseMoveEventListener und IMouseScrollEventListener

Grund: Die in IMouseEventListener definierten Methoden wurden in drei Interfaces aufgeteilt, damit nicht jede Klasse, die Teile der Funktionalität benötigt, alle Methoden überschreiben muss und die dann teilweise leer sind. Zudem hatten wir keinen MoveEventListener, was allerdings nötig war.

3.2 GameObjects:

3.2.1 EdgeColoring

Grund: War im Entwurf nicht spezifiziert, da die Einfärbung als optional angesehen wurde. Ein einfacher Inputhandler analog zu EdgeRectangles, der auf Tastendruck einen ColorPickDialog für die selektierten Kanten öffnet.

3.2.2 EdgeRectangles

Grund: War im Entwurf nicht spezifiziert, da das Anlegen von Flächen als optional angesehen wurde. Ein einfacher Inputhandler analog zu EdgeColoring, der auf Tastendruck den selektierten Kanten eine gemeinsame neue Flächen-ID zuweist.

3.2.3 ModelColoring

Grund: Alle GameModels halten jetzt nicht mehr nur ein Color-Objekt, das ihre Farbe beinhaltet, sondern eine Instanz von ModelColoring. Dies ist eine abstrakte Klasse, die die Farbe von Modellen kapselt. Konkrete Implementierungen dieser Klasse sind SingleColor für eine einheitliche Farbe und GradientColor für einen

Farbverlauf zwischen zwei Farben. Derzeit können wir aufgrund der Eingeschränktheit der XNA-API noch keinen Farbverlauf auf Modellen darstellen, aber sollten wir dazu noch eine Lösung finden, haben wir dann bereits die benötigte Infrastruktur im Code, um Modelle mit verschiedenen Farbeffekten zu versehen. Zusätzlich wird in ModelColoring auch die Hervorhebungsfarbe und dessen Intensität gekapselt und es gibt verschiedene Properties, um entweder die einzelnen Farbparameter abzufragen oder für Shadereffekte, die keine Farbverläufe unterstützen, eine Mischfarbe zu berechnen.

3.2.4 SkyCube

Grund: Ein IGameObject, das den weltraumartigen Hintergrund in unserem aktuellen Design darstellt.

3.2.5 TexturedRectangle

Grund: Ein texturiertes, frei im Raum positionierbares Rechteck, das aus zwei Dreiecken zusammengesetzt ist und low level gezeichnet wird.

3.2.6 TexturedRectangleInfo

Grund: Das Metadaten-Objekt zu TexturedRectangle.

3.3 RenderEffects:

3.3.1 OpaqueEffekt

Grund: Ein zusätzlicher RenderEffekt, welcher im Entwurf nicht berücksichtigt wurde.

3.3.2 RenderEffectLibrary

Grund: Sammelt alle aktuell im Spiel verwendbaren RenderEffekte und bietet Schnittstellen an, um den aktuell vom Spieler im Optionsmenü ausgewählten Rendereffekt zu instanziieren.

3.3.3 IRenderEffectStack

Grund: Wir brauchen ein gemeinsames Interface für den echten RenderEffectStack und dessen Mock-Objekt bei den Unit-Tests.

3.4 KnotData:

3.4.1 DirectionHelper

Grund: Enthält Methoden rund um Direction.

3.4.2 RectangleMap

Grund: Analog zu NodeMap stellt RectangleMap eine Zuordnung zwischen den Kanten und den Flächen zwischen ihnen dar. Die der RectangleMap übergebenen Kanten enthalten je keine, eine oder mehrere Flächen-IDs, aus denen dann in dieser Klasse die Positionen und Ausmaße der Flächen berechnet werden.

3.5 Widgets:

3.5.1 Border

Grund: Zeichnet einen einfarbigen Rahmen mit einer definierbaren Breite um ein Widget.

3.5.2 Bounds

Grund: Eine verbesserte Neuimplementierung der Rectangle-Klasse aus XNA, die auf ScreenPoint-Objekten basiert und erweiterte Funktionalität bietet, darunter die prozentuale Skalierung auf die aktuelle Bildschirmgröße.

3.5.3 ChallengePauseDialog

Grund: Vorher war nur ein Pause-Dialog vorgesehen. Es hat sich herausgestellt, dass wir für die beiden Modi unterschiedliche Funktionalität benötigen. Im Fall des Challenge-Modus muss der Pause-Dialog die Zeit anhalten.

3.5.4 ColorPickDialog

Grund: Ein Dialog, der ein ColorPickItem enthält und zum Farben-auswählen aufgerufen wird.

3.5.5 Container

Grund: Container wurde in Menu umbenannt und VerticalMenu in Menu, weil dies die angedachte Funktionalität besser beschreibt.

3.5.6 CreativePauseDialog

Grund: Vorher war nur ein Pause-Dialog vorgesehen. Es hat sich herausgestellt, dass wir für die beiden Modi unterschiedliche Funktionalität benötigen.

3.5.7 ErrorDialog

Grund: Ein Dialog der Fehlermeldungen anzeigt.

3.5.8 Lines

Grund: Ein Widget, das die Linien zeichnet, die im HfG-Design vorgegeben waren und so aussehen wie im Mockup.

3.5.9 MenuEntry

Grund: Umbenannter MenuButton.

3.5.10 ScreenPoint

Grund: Eine verbesserte Neuimplementierung sowohl der Klassen Vector2 als auch Point aus dem XNA-Framework, die sich implizit in beide konvertieren lässt und an deren Stelle in der XNA-API verwenden lässt. Die X- und Y-Korrdinaten von ScreenPoint sind als Delegate implementiert, was eine verschachtelte Abhängigkeit zwischen mehreren ScreenPoints ermöglicht. Jeder ScreenPoint hält eine Referenz auf den IGameScreen, in dem er verwendet wird, und skaliert bei einem Cast in Vector2 oder Point auf den aktuell im GameScreen verwendeten Viewport. Diese Implementierung ist von der erweiterten Monogame-API inspiriert und behebt sowohl die Schwäche, dass zwischen zwei Points des XNA-Frameworks keine Rechenoperationen ausgeführt werden können als auch das Problem, dass beide keine Delegate-Werte unterstützen und damit keine Abhängigkeiten und komplexeren automatischen Berechnungen durchgeführt werden können.

3.5.11 State

Grund: Umbenannter ItemState, da jetzt alle Widgets (nicht nur MenuItems) Selected oder Hovered sein können.

3.5.12 TextItem

Grund: Wir haben festgestellt, dass wir keine Möglichkeit hatten reine Textkomponenten für Dialoge oder Menüs zu nutzen. Wir hätten die Klasse des Buttons verwenden können, aber dies wäre nicht die korrekte Verwendung der Klasse im Sinne des Entwurfs. Aus diesem Grund haben wir eine Klasse für reine Textdarstellung eingefügt.

3.6 Audio:

Im Entwurf wurde das Thema Audio nicht berücksichtigt, da wir dachten uns hierbei auf XNA zu verlassen und aus diesem Grund wurden folgende Klassen hinzugefügt.

3.6.1 AudioManager

Grund: Verwaltet alle Audio-Dateien die im Spiel verwendet werden.

3.6.2 IAudioFile

Grund:

3.6.3 IPlaylist

Grund:

3.6.4 LoopPlaylist

Grund:

3.6.5 OggVorbisFile

Grund:

3.6.6 Sound

Grund:

3.6.7 SoundEffectFile

Grund:

3.7 Utilities:

3.7.1 BoundingCylinder

Grund:

3.7.2 ColorHelper

Grund:

3.7.3 DictionaryHelper

Grund:

3.7.4 EnumHelper

Grund:

3.7.5 FileIndex

Grund:

3.7.6 FileUtility

Grund:

3.7.7 FrustumHelper

Grund:

3.7.8 HfGDesign

Grund:

3.7.9 IniFile

Grund:

3.7.10 InputHelper

Grund:

3.7.11 ModelHelper

Grund:

3.7.12 MonoHelperMG

Grund:

3.7.13 MonoHelperXNA

Grund:

3.7.14 RayExtensions

Grund:

3.7.15 SavegameLoader

Grund:

3.7.16 ShaderHelper

Grund:

3.7.17 TextHelper

Grund:

3.7.18 TextureHelper

Grund:

3.7.19 VectorHelper

Grund: