

ENTWURFSDOKUMENT

(V. 1.0)

KNOT³

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.-Inf. M. Retzlaff

Auftragnehmer:

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

11. Dezember 2013

Inhaltsverzeichnis

1	Einleitung	2
2	Aufbau	3
2.1	Architektur	3
2.2	Klassendiagramm	4
2.3	Verwendete Entwurfsmuster	4
3	Klassenübersicht	5
4	Abläufe	6
4.1	Sequenzdiagramme	6
5	Klassenindex	7
6	Anmerkungen	8
7	Gloassar	9

Kapitel 1

Einleitung

Das Knobel- und Konstruktionsspiel Knot³, welches im Auftrag des IBDS Dachsbacher ausgearbeitet wird, wird wie im Pflichtenheft spezifiziert angefertigt.

Kapitel 2

Aufbau

2.1 Architektur

Die grundlegende Architektur des Spiels basiert auf der Spielkomponenten-Infrastruktur des XNA-Framework, die mit Spielzuständen kombiniert wird. Die abstrakten Klassen `GameStateComponent` und `DrawableGameStateComponent` erben von den von XNA bereitgestellten Klassen `GameComponent` und `DrawableGameComponent` implementieren zusätzlich die Schnittstelle `IGameStateComponent`. Sie unterscheiden sich von den XNA-Basisklassen dadurch, dass sie immer eine Referenz auf einen bestimmten Spielzustand halten und nur in Kombination mit diesem zu verwenden sind.

Die Spielzustände erben von der abstrakten Basisklasse `GameScreen` und halten eine Liste von `IGameStateComponent`-Objekten. Wird ein Spielzustand aktiviert, indem von einem anderen Spielzustand aus zu ihm gewechselt wird oder indem er der Startzustand ist, dann weist er seine Liste von `IGameStateComponent`-Objekten dem `Components`-Attribut der `Game`-Klasse zu, die von der vom XNA-Framework bereitgestellten abstrakten Klasse `Game` erbt. So ist zu jedem Zeitpunkt während der Laufzeit des Spiels ein Spielzustand aktiv, der die aktuelle Liste von Spielkomponenten verwaltet.

Die Spielkomponenten, die nicht gezeichnet werden und nur auf Eingaben reagieren, haben nur eine `Update()`-Methode und erben von `GameStateComponent`. Dies sind vor allem verschiedene Input-Handler, die Tastatur und Mauseingaben verarbeiten und die beispielsweise die Kameraposition und das Kameratarget ändern oder Spielobjekte bewegen.

Spielkomponenten, die neben der `Update()`-Methode auch eine `Draw()`-Methode besitzen, erben von `DrawableGameStateComponent`. Dies sind vor allem die Elemente, aus denen die grafischen Benutzeroberfläche zusammengesetzt ist, deren abstrakte Basisklasse `Widget` darstellt. [weitere Erklärungen zu Widgets...]

Alle Spielobjekte implementieren die Schnittstelle `IGameObject`. Die abstrakte Klasse `GameModel` repräsentiert dabei ein Spielobjekt, das aus einem 3D-Modell besteht, und hält zu diesem Zweck eine Referenz auf ein Objekt der Klasse `Model` aus dem XNA-Framework sowie weitere Eigenschaften wie Position, Drehung und Skalierung.

Spielobjekte sind keine Komponenten, sondern werden in einer Spielwelt zusammenfasst, die durch die Klasse `World` repräsentiert wird. Die Spielwelt ist ein `DrawableGameStateComponent` und ruft in ihrer `Update()`- und `Draw()`-Methoden jeweils die dazugehörigen Methoden aller in ihr enthaltenen Spielobjekte auf.

Shadereffekte werden durch die abstrakte Klasse `RenderEffect` und die von ihr abgeleiteten Klassen gekapselt. Ein `RenderEffect` enthält ein Rendertarget vom Typ `RenderTarget2D` als Attribut und implementiert jeweils eine `Begin()`- und eine `End`-Methode. In der Methode `Begin()` wird das aktuell von XNA

genutzte Rendertarget auf einem Stack gesichert und das das Rendertarget des Effekts wird als aktuelles Rendertarget gesetzt.

Nach dem Aufruf von `Begin()` werden alle `Draw()`-Calls von XNA auf dem gesetzten Rendertarget ausgeführt. Es wird also in eine im `RenderTarget2D`-Objekt enthaltene Bitmap gezeichnet. Dabei wird von den `Draw()`-Methoden der `GameModels` die `DrawModel(GameModel)`-Methode des `RenderEffects` aufgerufen, der die Modelle mit bestimmten Shadereffekten in die Bitmap zeichnet.

In der `End()`-Methode wird schließlich das auf dem Stack gesicherte vorher genutzte Rendertarget wiederhergestellt und das Rendertarget des `RenderEffects` wird, unter Umständen verändert durch Post-Processing-Effekte, auf dieses übergeordnete Rendertarget gezeichnet.

2.2 Klassendiagramm

2.3 Verwendete Entwurfsmuster

Kapitel 3

Klassenübersicht

Kapitel 4

Abläufe

4.1 Sequenzdiagramme

Kapitel 5

Klassenindex

Kapitel 6

Anmerkungen

Kapitel 7

Gloassar