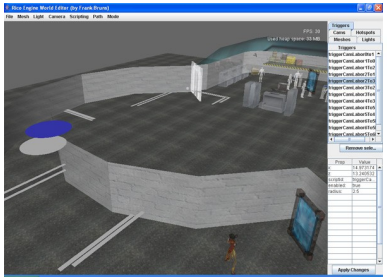
Frank Bruns
15.01.2008

Präsentation der Master-Arbeit von Frank Bruns



"Entwicklung eines Java basierten Frameworks für 3D Point & Click Adventures"

Bearbeitungszeitraum: 06.06.2007 - 06.12.2007

Die Aufgabenstellung

- Prototypische Implementierung eines Systems zur Erstellung von 3D Point & Click Adventures

- Grundlegende technische Vereinbarungen dazu:
  - Verwendung der Programmiersprache **Java**
  - 3D-Darstellung mit **OpenGL** (eigentlich ein API für C)

  → **JOGL**-API als Brücke zwischen Java und OpenGL

Präzisierung der Aufgabenstellung

1. Implementierung einer 3D-Game-Engine zur Darstellung und Transformation von Spielwelten

2. Implementierung eines auf Adventure-Spiele ausgerichteten Frameworks
   → soll auf die 3D-Engine aufgesetzt werden

3. Umsetzung eines Demo-Spiels auf Basis des entwickelten Systems

Persönliches Ziel der Arbeit

Ich will zeigen:

Java eignet sich doch für die Spiele-Programmierung!

Was sind Point & Click Adventures?

- Sammelbegriff für ein Genre von Video-Spielen
- Zentrale Merkmale:
  - Steuerung i.d.R. nur mit der Maus (daher Point & Click)
  - Spielwelt ist in diskrete Räume (Locations) gegliedert
  - Interaktion nur an vordefinierten Punkten (Hotspots)
  - Einsammeln von Items und Führen von Unterhaltungen
  - Lösen von Rätselaufgaben
    → Kombination der Inventargegenstände mit Hotspots und Bezugnahme auf Gesprächsinformationen
  - Schwerpunkt auf den narrativen Elementen (fesselnde Story)

Beispiele für Point & Click Adventures

- 2D basiert:
  - Monkey Island 1-3, Day of the Tentacle
  - Runaway 1 & 2, Baphomets Fluch 1 & 2
- Hybrid-Ansatz (2.5D) → aktuell oft verwendet:
  - The Moment of Silence, Geheimakte Tunguska
- Vollständiges 3D:
  - Ankh 1-3, The Westerner, Jack Keane
  - Mein Demo-Spiel!

Vorgehen bei der Entwicklung

- Drei diskrete Projektabschnitte
  - 1. Entwicklung der (generischen) 3D-Engine
  - 2. Entwicklung des Adventure-Frameworks
  - 3. Umsetzung des Demo-Spiels

- Natürlich erfolgten in der Praxis gelegentlich Veränderungen/Verbesserungen am Code des vorherigen Projektabschnitts

1. Projektabschnitt

Entwicklung der 3D-Engine

Wichtige Anforderungen an die Engine (1/2)

- Eigenständigkeit
  - Unabhängigkeit von der Adventure spezifischen Schicht
- Objektorientiertheit
  - Vorteile der OO-Programmierung nutzen
- Performanter Render-Prozess
  - Einsatz selektiver Render-Techniken
- Import (animierter) 3D-Modelle
  - Model-Loader für geeignete 3D-Dateiformate
- Kamera-System
  - Dynamische Veränderungen des Blickwinkels

Wichtige Anforderungen an die Engine (2/2)

- Kollisionserkennung
  - Ressourcen schonende Kollisionstests
- Wegfindungssystem
  - Start-Ziel-Pfadberechnungen um Hindernisse herum
- Optische Spezialeffekte
  - Visuell ansprechendere, realistischere Spielwelten
- Sound-System
  - Wiedergabe (un-)komprimierter Audio-Dateien
- Unterstützung peripherer Eingabegeräte
  - Verarbeitung von Maus- und Tastatureingaben

Umsetzung der Anforderungen (1/6)

- Eigenständigkeit
  - Implementierung zeitlich vor der Adventure Schicht
- Objektorientiertheit
  - Kapselung imperativen OpenGL-Codes in OO-Strukturen
  - Modularität durch striktes OO-Klassendesign
  - Erweiterbarkeit durch abstrakte Klassen und Interfaces
- Performanter Render-Prozess
  - Hardware beschleunigtes Rendern dank OpenGL
  - Frustum-Culling → nur Objekte im Sichtkegel zeichnen
  - Occlusion-Culling → nur unverdeckte Objekte zeichnen

Umsetzung der Anforderungen (2/6)

- Import (animierter) 3D-Modelle
  - Weiterentwicklung des **MD2**-Loaders aus der JOGL-Vorlesung
  - Adaption des freien **3DS**-Loaders des joglutils-Projekts
  - Adaption eines freien **MilkShape3D**-Loaders aus dem Netz

- Kamera-System
  - Global zugängliche Singleton-Instanz
  - Frei beweglich in der 3D-Umgebung
  - Zuweisung vordefinierter Kamera-Perspektiven möglich
  - Aufzeichnung und Wiedergabe von Kamera-Fahrten

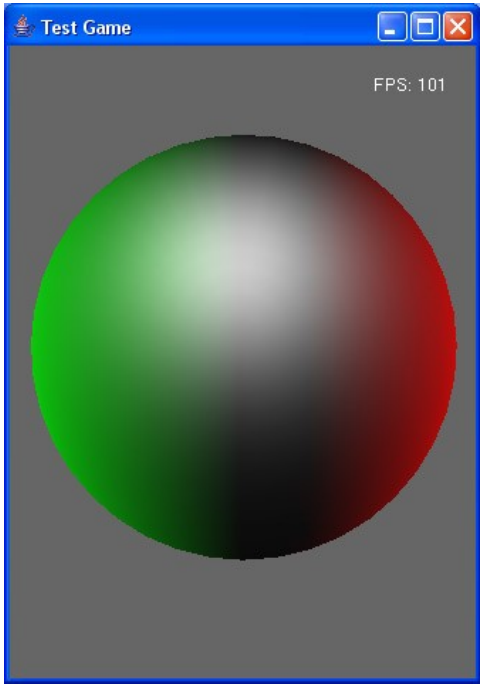
Umsetzung der Anforderungen (3/6)

- Kollisionserkennung
  - Performante Kollisionstests durch Bounding Volumes
    → jedoch weniger genau als Triangle basierte Kollisionstests
  - Test auf Kollisionen zwischen Bounding Spheres
  - Test auf Kollisionen zwischen Bounding Boxes

- Wegfindungssystem
  - Verwendung des (heuristischen) A*-Algorithmus
  - Ergebnis-Pfad hängt von Güte der verwendeten Heuristik ab
  - Wegfindung anhand eines Bodengitters (Grid), für das vorab passierbare und unpassierbare Bereiche definiert werden

Umsetzung der Anforderungen (4/6)

- Optische Spezialeffekte 1

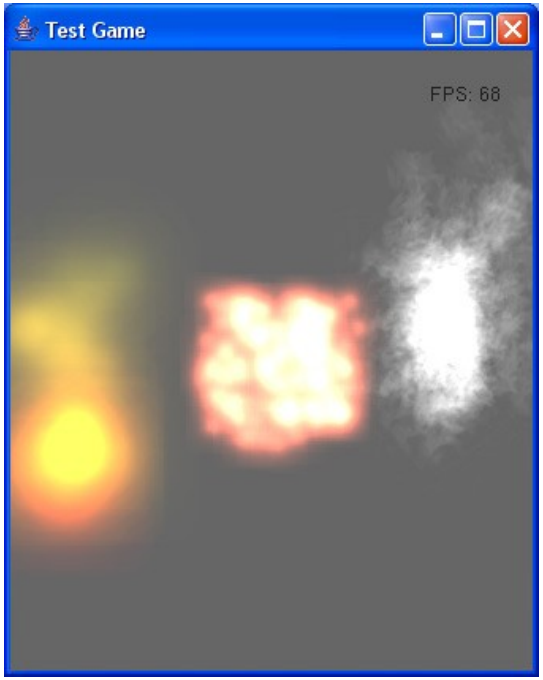


Beleuchtungssystem

Dynamischer Schattenwurf

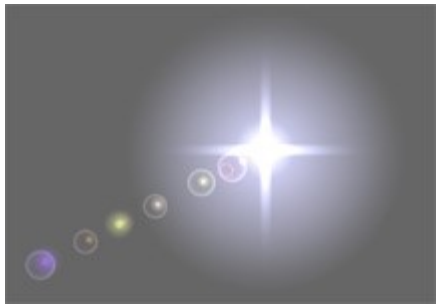
Umsetzung der Anforderungen (5/6)

- Optische Spezialeffekte 2



Partikel-System

Lens-Flare



Environment Mapping

Umsetzung der Anforderungen (6/6)

- Sound-System
  - Kapselung von Sounds in Objekte spezieller Klassen
  - Gemeinsame Interface-Methoden
    → Handhabung der Audio-Formate identisch
  - Unterstützt das WAV-, MIDI- und MP3-Format
- Unterstützung peripherer Eingabegeräte
  - Input-System, das sowohl Maus- als auch Tastatureingaben komfortabel verarbeitet
  - Mauszeiger in der Bildmitte fixierbar
    → Voraussetzung für Mouse-Look-Modus

2. Projektabschnitt

Entwicklung des Adventure-Frameworks

Wichtige Anforderungen an das Framework (1/3)

- Aufsatz auf meine 3D-Engine
- Unterstützung diverser Szene-Komponenten
  - Nicht-interaktive Kulissen-Elemente
  - Items
  - Charaktere (Spieler und NPCs)
  - Virtuelle Hotspots
  - Trigger
- Dateiformat zur Spezifikation von Räumen
  - Ermöglichung der persistenten Speicherung der Konstruktionsvorschrift eines Raums

Wichtige Anforderungen an das Framework (2/3)

- Sequenziell ausführbare Spielaktionen
  - **Beispiel:** Befehl zum Aufnehmen eines Item
    → Figur erst zum Item laufen lassen, dann Item nehmen
- Inventar-System
  - Verwaltung eingesammelter Items
- Dialog-System
  - Möglichkeit Unterhaltungen mit NPCs zu führen

Wichtige Anforderungen an das Framework (3/3)

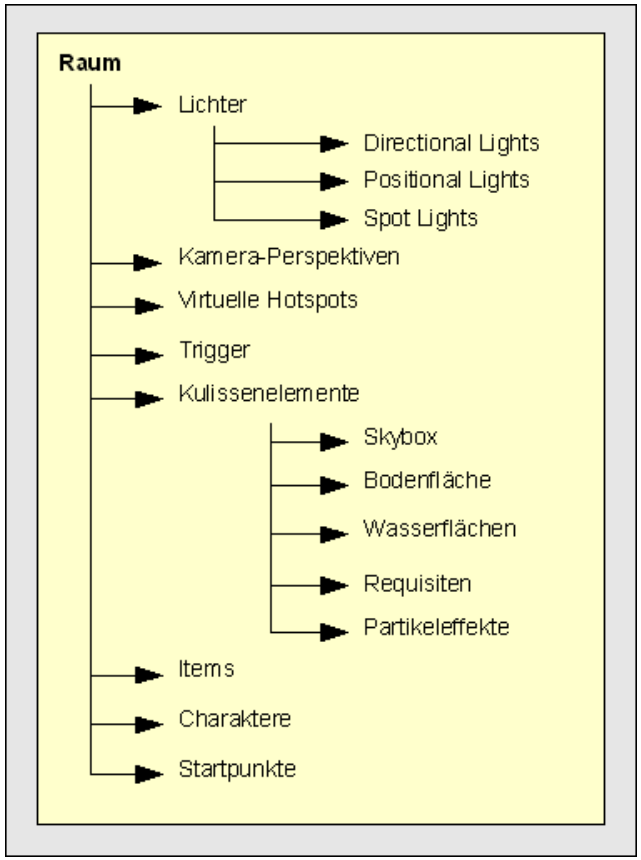
- Intuitives Kommando-Interface
  - Einfache, schnell erlernbare Spielsteuerung
  - Orientierung an den Konventionen des Genres
- Game-Scripting
  - Externe Skripte zur Definition der Spiel-Logik

Umsetzung des Frameworks (1/7)

- Aufbau der Spielwelt
  - Unterteilung in Kapitel, die einzelne Räume enthalten
  - Räume enthalten alle interaktiven und nicht-interaktiven Elemente aus den Anforderungen
  - Virtuelle Hotspots markieren einen interaktiven Bereich durch unsichtbare Kugeln oder Quader
  - Trigger bestehen aus unsichtbaren, kontaktsensitiven Kreis- oder Rechtecksflächen auf der Bodenebene

Umsetzung des Frameworks (2/7)

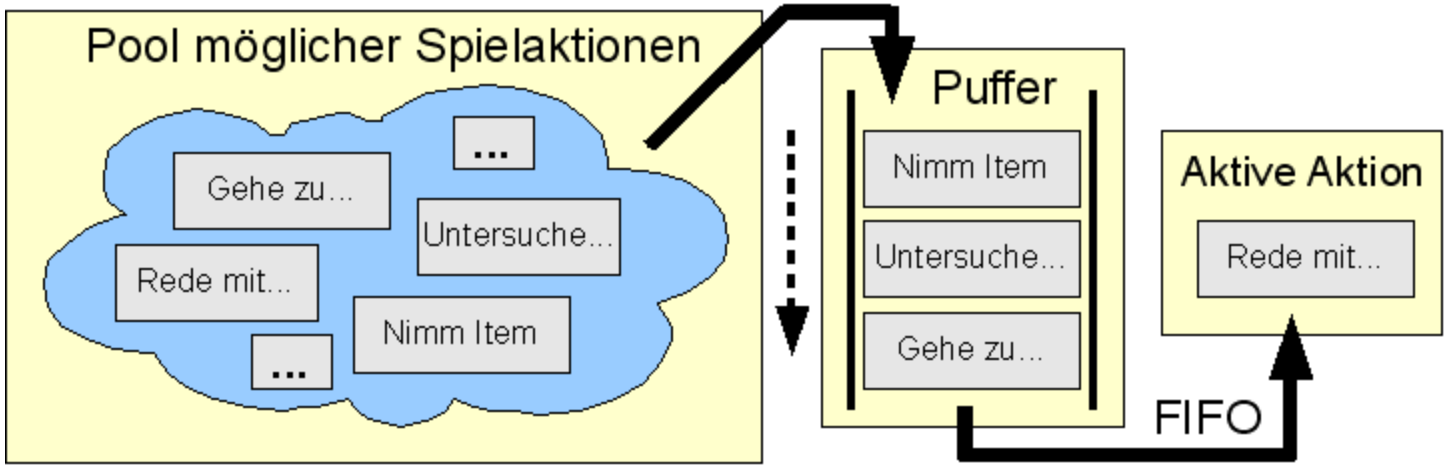
- Dateiformat zur Spezifikation von Räumen



Persistente Speicherung als XML-Dokument mit der Datei-Endung “.room”

Umsetzung des Frameworks (3/7)

- Sequenziell ausführbare Spielaktionen
  - Konzept des Action-Schedulers
    - Verwendet eine `LinkedList` als FIFO-Struktur

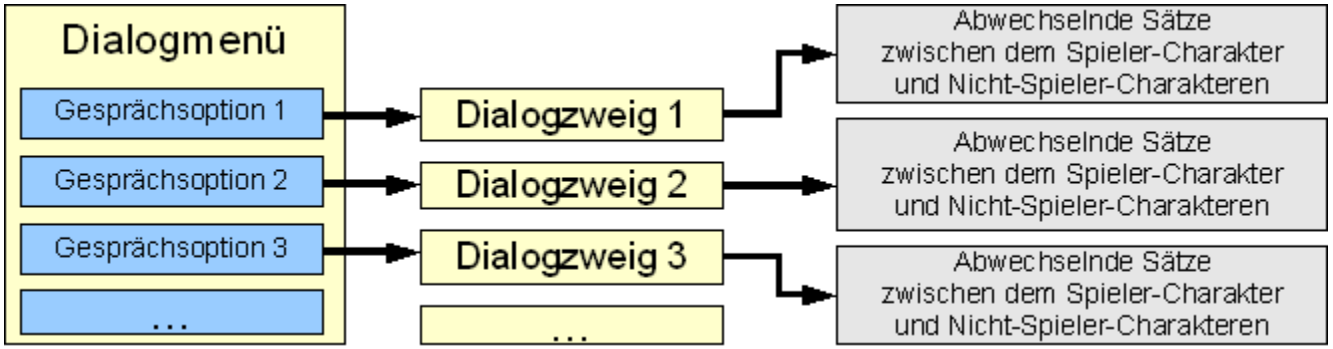


Umsetzung des Frameworks (4/7)

- Inventar-System
  - Umsetzung eines Balkeninventars
  - Durch abstraktes Klassendesign leicht um andere Varianten (z.B. Kisten-Inventar) erweiterbar

Umsetzung des Frameworks (5/7)

- Dialog-System
  - Bietet Auswahl verschiedener Gesprächszweige
  - Dialoge werden ebenfalls in XML-Dateien spezifiziert



Umsetzung des Frameworks (6/7)

- Mechanik der Spielsteuerung
  - Rechtsklick: Untersuche einen Hotspots
  - Mittlere Maustaste: Anzeige der Hotspots
  - Linksklick: kontextsensitive Aktion (siehe Tabelle)

Hotspot-Typ	Mausklick ohne gewähltes Inventar-Item	Mausklick mit gewähltem Inventar-Item
Item	Item einstecken	-- Keine Aktion vorgesehen --
Nicht-Spieler-Charakter	Konversation starten	Benutze Item mit dem Nicht-Spieler-Charakter
Virtueller Hotspot	Benutze Hotspot (gegebenenfalls nicht ohne gewähltes Item möglich)	Benutze Item mit Hotspot

Umsetzung des Frameworks (7/7)

- Game-Scripting
  - Verwendung der Skript-Sprache BeanShell
  - Eine Skript-Datei pro Raum definiert Reaktionen auf die Aktionen des Spielers
    - Spezifikation der Logik der interaktiven Elemente
      → **Trennung des Spiel-Inhalts vom Framework**
  - Gliederung der Skript-Datei in einzelne Methoden, die einer festen Namenskonvention folgen
  - Auf alle relevanten Spielobjekte kann in den Skripten manipulativ zugegriffen werden → **äußerst mächtig!**

3. Projektabschnitt

Umsetzung des Demo-Spiels

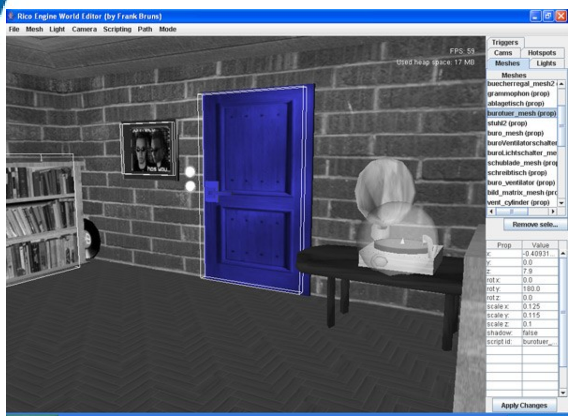
Die Spielidee in groben Zügen

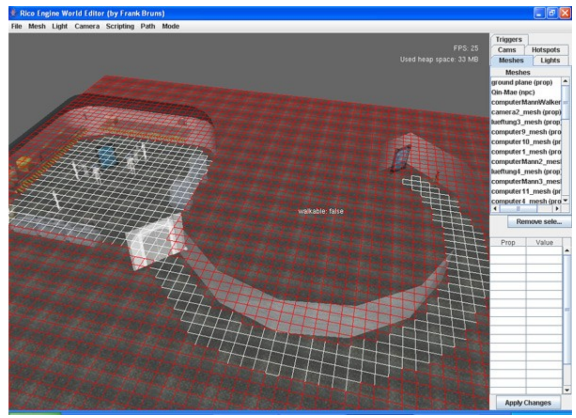
- Zeitreise-Geschichte
  - Spielfigur muss seinen Vater aus dem Mittelalter retten
- Spieler besucht verschiedene Epochen
  - Chicago im Jahre 1931: Die Gegenwart des Spielers
  - Ferne Zukunft (2369)
  - Mittelalter
- Unterschiedliche Farbgebung in den Epochen
  - Mittelalter: Sepia-Tönung
  - 1931: Schwarz-Weiß
  - Zukunft: korrekte bunte Farbgebung

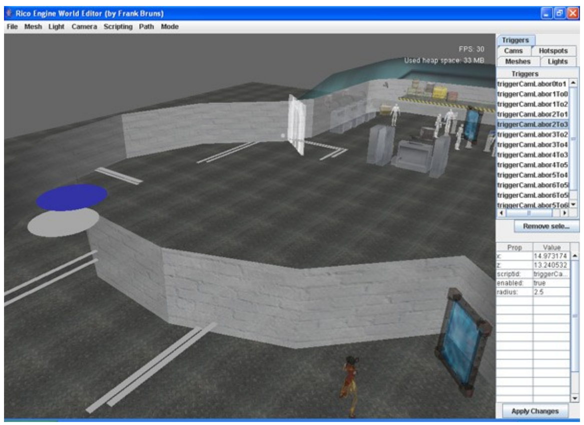
Frank Bruns
15.01.2008

Hintergrundwissen zum Gesamtprojekt

- Ein kleiner 3D-Raum-Editor als „Nebenprodukt“







- Lines of Code (gesamt): ca. 50.000

- Anzahl Java-Klassen (gesamt): ca. 250

Den Worten sollen Taten folgen!

Vorführung des Demo-Spiels

Das war's schon fast

Vielen Dank für die Aufmerksamkeit.

Jetzt stehe ich gerne für Fragen zur Verfügung :-)