Projekt "teddys"

Sebastian Kahl, Christian Mücke, Benjamin Siemoneit

Game Engineering und Simulation
Sommersemester 2012

Universität Bielefeld

Thema / Idee

- Sparse Network Multiplayer Game
- Anlehnung an Worms
- Die natürliche Umgebung eines mit Jetpack ausgestatteten Teddys ist gespickt mit Waffen. Weil Teddys Einzelgänger sind, können Konflikte mit anderen Artgenossen nicht ausgeschlossen werden.

Technischer Hintergrund

- Programmiersprachen: Java, XML, "Shader"
  - Java: Spiellogik, Netzwerkeinheit
  - XML: Level-Beschreibungen, GUI-Elemente
  - Shader: Grafische (Bewegungs-)Abläufe
- Framework: jMonkeyEngine 3.0.0 beta
  - ebenfalls in Java geschrieben
- Entwicklung unter MacOS und Linux
- Versionierung: Git
- Kommunikation: Google Docs, Google Groups, Google Projects, Git-Commit-Messages, Reale Welt

jMonkeyEngine

- Vorteile
  - Kostenlos
  - Einfacher Einstieg, gute Community
  - Umfasst viele vordefinierte Konzepte
  - Integrierte IDE (Netbeans-Derivat)
  - Plattformunabhängig (Java+OpenGL)
  - Gute Skalierung mit #CPU
- Nachteile
  - Beta-Status bemerkbar
  - Teilweise umständliche Umsetzung von Zielen
  - Teilweise unvollständige Doku
  - Unvollständiger Import von Blender-Modellen

Besonderheiten

- Größeres Projekt
- Neue Aspekte der Anwendungsentwicklung
  - Grafische Modelle
  - Netzwerkkommunikation
  - Threading-Konzepte in Verbindung mit Szenegraphen
  - Integration verschiedener Bereiche im Build
- Gutes Zusammenspiel verschiedener Tools
- Es hat Spaß gemacht

Probleme

- Features
  - Fehleinschätzung der zeitlichen Umsetzung
  - Framework teilweise umständlich
- Framework
  - Eigenarten müssen nachvollzogen werden
- Java
  - U.U. Lange Stacktraces
  - Debugging von Threads
- Server/Client-Konzept
  - Umfangreiche Abhängigkeiten in der Logik
  - Anwendungsverhalten ausgehend von Nachrichten

Projektplan

- Geplantes Vorgehensmodell: Iterativ-inkrementell
  - Gut zur Anforderungserhebung
  - Aber: Risikofaktoren wurden "unwichtig" ...
  - Abkehr von festen Zeitframes durch Zeitprobleme
- Umgesetztes Vorgehensmodell: Kontinuierliche Integration
  - Hier: Keine automatischen Builds, keine Unit-Tests
  - Vorteil: Kurze Entwicklungszyklen
  - Unterstützt durch Git-Versionierung
  - Nachteil: Selbstständiges Debugging

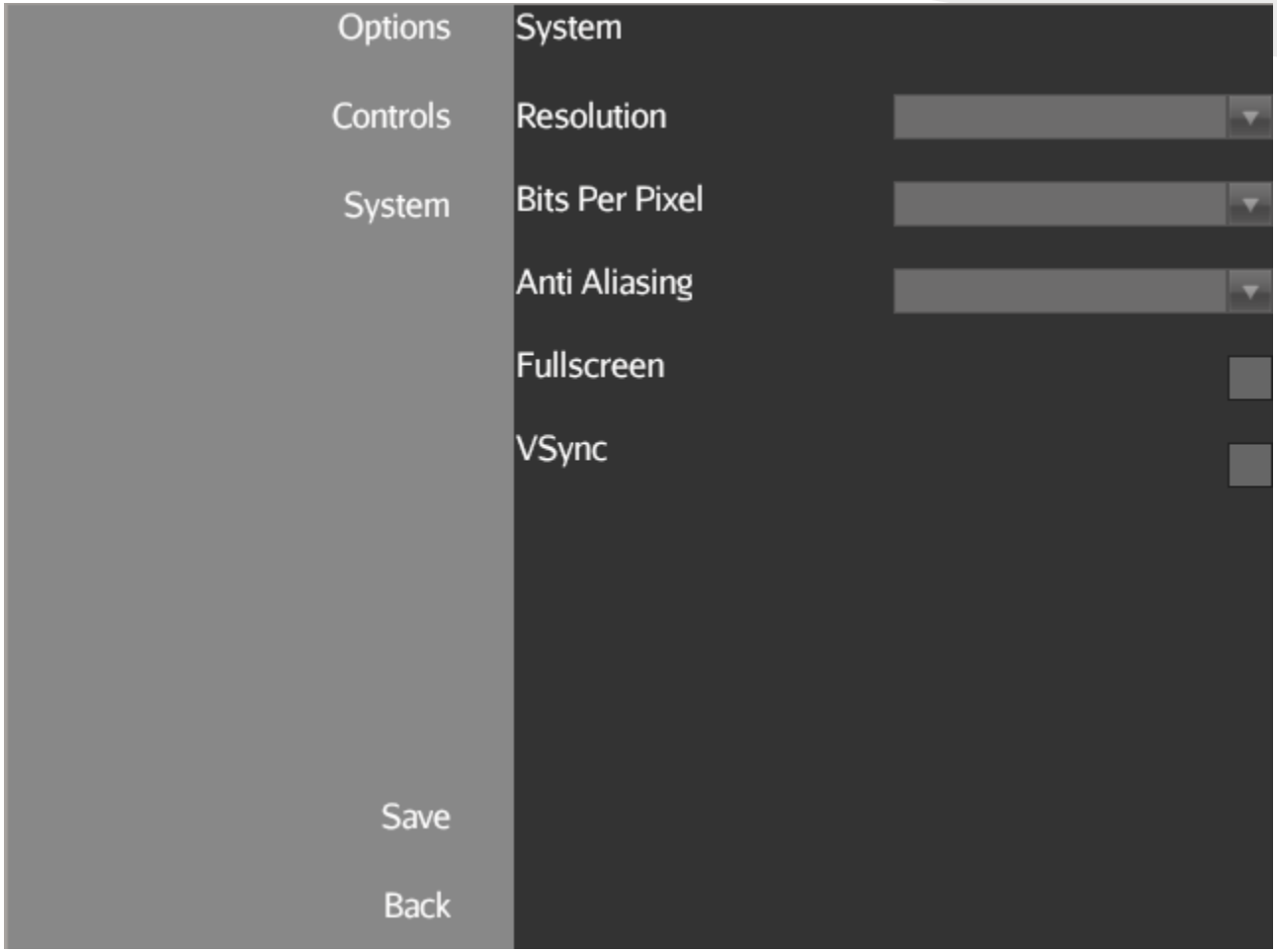
Verantwortlichkeiten

- Benjamin: Menüs, HUD, Perspektive, Szenegraph
- Christian: Netzwerk, Steuerung
- Sebastian: Grafik (Animationen, Texturen, Shader, Effekte), Spiellogik

Hauptmenü

Hauptmenü

Optionsmenü



Joinmenü