

**Jump-And-Run**

**Software-architektur Dokument**

**Version 3.0**

**Überarbeitungshistorie**

|  |  |  |  |
| --- | --- | --- | --- |
| **Datum** | **Version** | **Beschreibung** | **Autor** |
| 15.06.2013 | 1.0 | Formatvorlage | Timo Krauskopf |

**Inhaltsverzeichnis**

[1. Einleitung 4](#_Toc413326120)

[1.1. Zweck 4](#_Toc413326121)

[1.2. Geltungsbereich 4](#_Toc413326122)

[1.3. Definitionen, Akronyme, und Abkürzungen 4](#_Toc413326123)

[1.4. Referenzen 5](#_Toc413326124)

[2. Darstellung der Architektur 5](#_Toc413326125)

[3. Ziele und Einschränkungen der Architektur 5](#_Toc413326126)

# Einleitung

## Zweck

Das Dokument bietet einen Überblick über die Architektur der Software Jump-and-Run an Hand verschiedener Sichten für die verschiedenen Aspekte des Systems.

Signifikante Architektur Entscheidungen werden erläutert. Ziel ist es dabei, wichtige Entscheidungen in der Design-Phase gründlich zu überdenken, damit in der Realisierungsphase möglichst wenig Komplikationen auftreten.

## Geltungsbereich

PC Software für das Spiel Jump-and-Run

## Definitionen, Akronyme, und Abkürzungen

Siehe Glossar

## Referenzen

* + - Vision
    - Software Anforderungs Spezifikation
    - MS Project Iteration Plan
    - Glossar

# Darstellung der Architektur

Um das Softwaresystem so exakt wie möglich darstellen zu können, wird zuerst Kontexterzeugend hierarchisch vorgegangen, danach werden die Komponenten und die Relationen mit dem nötigen Detail beschrieben.

Die Diagramme werden mit dem Programm Visual-Paradigm oder dem Internet Service von Creately in UML-konformer Notation erstellt.

Das Dokument beschreibt den Aufbau des Softwaresystems. Es wurde hierbei nicht nach einem externen Muster Vorgegangen, sondern wird vom Grobaufbau immer mehr ins Detail gegangen.

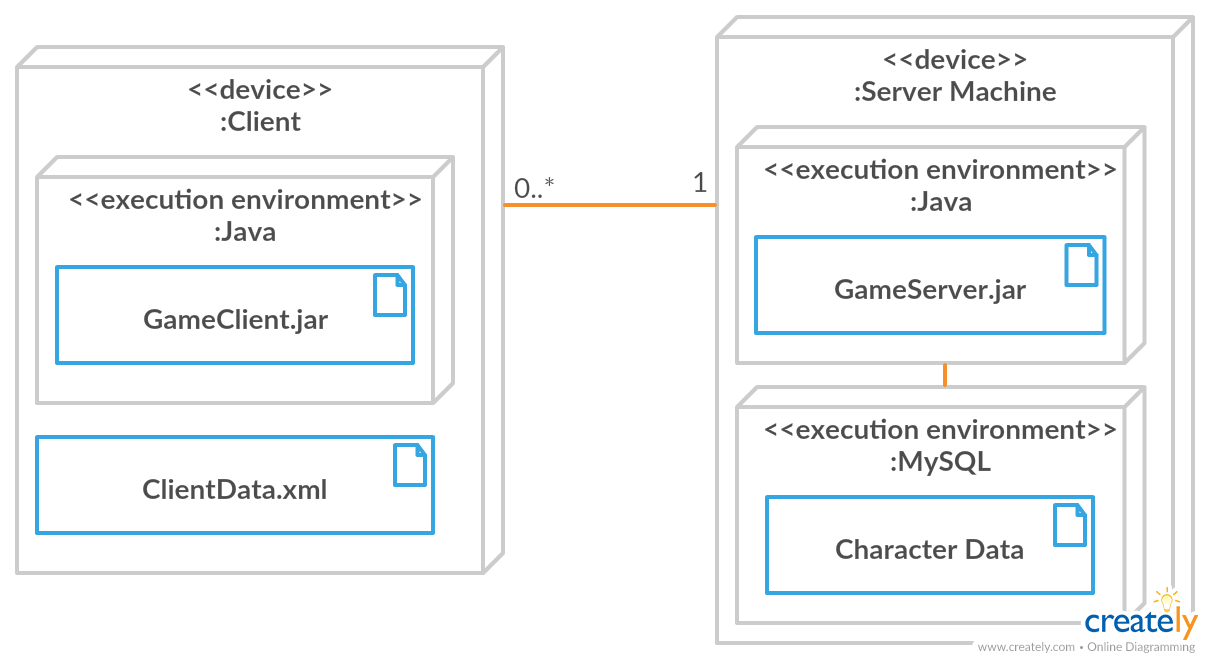
Innerhalb des Dokuments wird nur die Architektur der Software Jump-and-Run Spiel für PC-Anwender mit Serveranbindung sowie die Architektur des Servers beschrieben.

# Ziele und Einschränkungen der Architektur

Im Folgenden werden diejenigen Anforderungen aufgelistet, welche einen bedeutsamen Einfluss auf die Software Architektur haben.

Die Architektur soll so gestaltet werden, dass ein modulares System mit austauschbaren Komponenten entsteht.

# Verteilungsdiagramm



Bei der Software Jump-and-Run Spiel wird, um das Zusammenspielen zwischen Spielern zu ermöglichen, mit einer Client-Server Architektur gearbeitet. Für das speichern Persistenter Daten auf dem Server wird eine SQL Datenbank auf der Basis von MySQL verwendet. Sie soll Daten über Benutzer und deren Charakter speichern.

Sowohl auf dem Server sowie auf dem Client wird eine Java Software laufen. Auf dem Client werden persistente Daten unterschiedlich gespeichert. Die meisten Daten davon werden in xml Dateien gespeichert.

Jedoch werden einzelne Daten in noch nicht definierten anderen Formaten gespeichert um dem Benutzer die Möglichkeit zu nehmen auf diese einfach zuzugreifen.

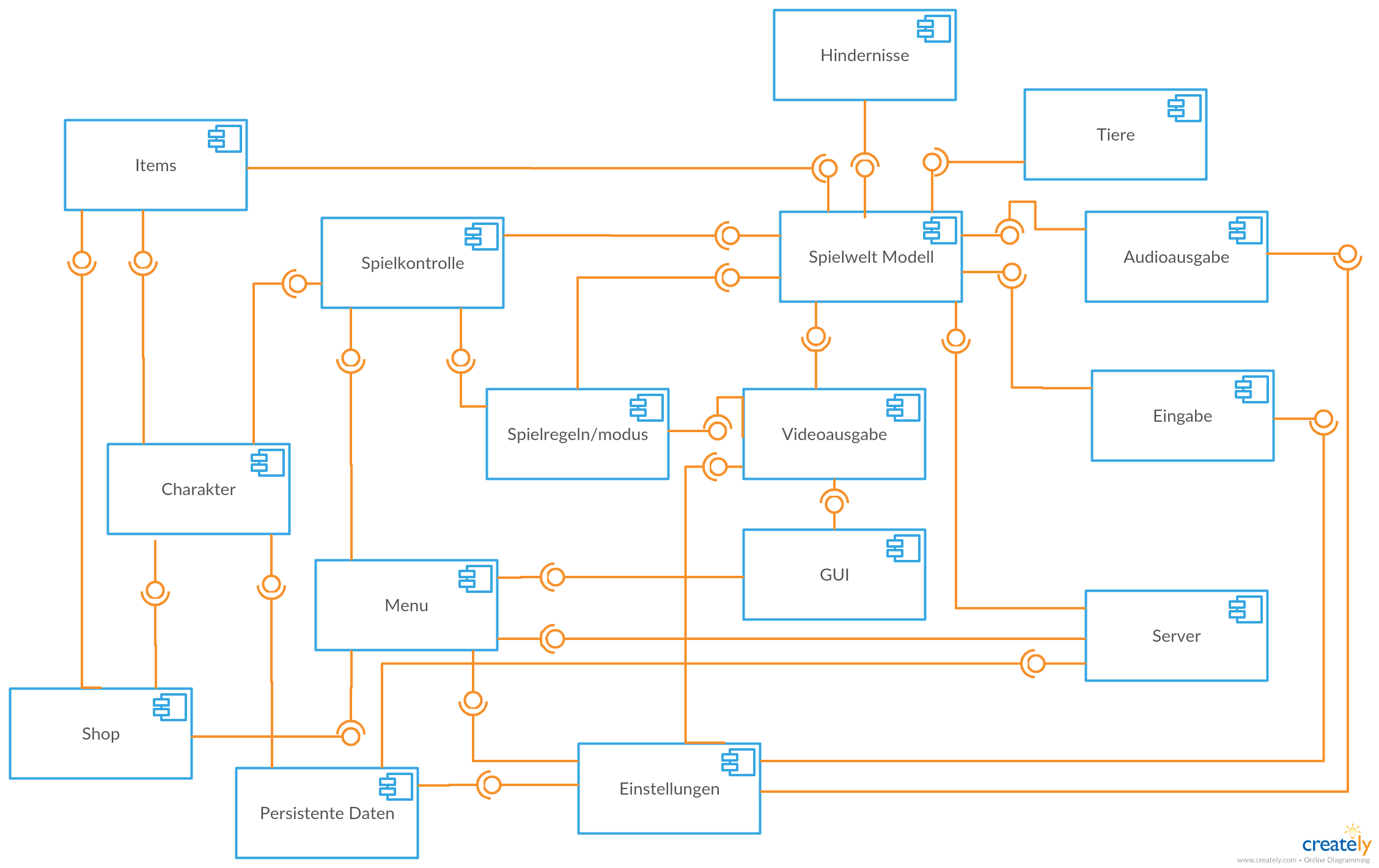
# Clientarchitektur

# Package Diagram

# 

Bei dem Spielclient planen wir 3 große Pakete ein sowie 3 kleinere Pakete, von denen 2 abgespalten wurden um den Code einfacher zu verwalten. Die großen Pakete sind die Server Kommunikation der Spielkern sowie die Menü und GUI Elemente. Sie umfassen eigentlich die ganze Softwarelogik des Spielclients. Spieleinausgabe sowie Character und Besitz wurden von GUI und Spielkern abgespaltet um eine zusätzliche Barriere zwischen diesen Paketen zu errichten. Das Paket PersistenData hat keinen klaren Besitzer. Es regelt das laden und speichern sowie updaten Persistenter Daten.

# Component Diagram



**Spielwelt Modell**

stellt den Kern des Spiels dar. In ihm werde alle Spielelemente gesammelt welche innerhalb eines Spiels vorhanden sind. Alle Physikalischen Berechnungen finden hier statt. Zudem werden hier alle Spielaktionen ausgeführt. Desweiteren findet man hier auch die Levelgeneration. Externe Komponenten können auf diese Komponente zugreifen um die Spiellogik zu manipulieren, Teil davon zu werden oder diese zu überwachen.

**Spielkontrolle**

liefert Kontrolle über den Spielzustand in der simpelsten Form. Es startet ein neues Spiel mit bestimmten Einstellungen und pausiert oder beendet ein laufendes. Zudem informiert es über Ergebnisse beendeter Spiele.

**Spielregeln/modus**

eigentlich ein Bestandteil der Spielwelt wurde sie aus diesem Extrahiert um sie einfacher getrennt zu bearbeiten. Er liefert zusätzliche Informationen an die Videoausgabe welche für den Modus spezifisch sind – und beeinflusst das Spielgeschehen um die Spielregeln durchzusetzen. Dafür überwacht es die Spielwelt mit einem Observer pattern.

**Items, Tiere, Hindernisse**

exportieren genannte Spielelemente in die Spielwelt. Sind alles seperate Komponenten um sie leichter zu Verwalten. Sie erhalten ihren Effekt durch den Eingriff in die Spielwelt.

**Charakter**

mehr als nur der Spielcharakter, bietet Benutzerkontrolle, speichert die Highscores sowie den Besitz.

Wir persistent gespeichert und eventuell mit dem Server Synchronisiert und von diesem Validiert.

**Shop**

Der Itemshop erlaubt es Items mit der Spielwährung zu kaufen, sodass diese dann in den Besitz des Charakters übergehen. Wird eventuell vom Server validiert.

**Menu**

enthält die Menüführung des Spiels und exportiert eine Schnittstelle um weitere Menümasken hinzuzufügen und diese einheitlich Darzustellen. Dazu wird es auf die GUI Komponente zugreifen und diese Manipulieren.

Des weiteren sammelt es in verschiedenen Menümasken Informationen über verschiedene Elemente und zeigt diese dem Nutzer an.

**Einstellungen**

sammelt die Einstellungen des Spiels. Hauptsächlich sind dies die Einstellungen über Eingabe sowie die Audio- und Videoausgabe.

**Persistente Daten**sammelt die Persistente Daten des Spiels und schreibt sie in ihre entsprechenden Module – namentlich Charakter und Einstellungen. Die Daten können entweder vom Server oder von der Klientenmaschine holen.

**GUI**

eröffnet dem Menu und der Videoausgabe Zugriff auf eine grafische Benutzeroberfläche und liefert Funktionalität um diese zu verwalten und zu manipulieren.

**Videoausgabe**

stellt die Spielwelt sowie die Erweiterungen durch die verschiedenen Spielmodi dar. Verwendet dafür ein Observer Pattern mit dem die Spielwelt und der Spielmodus überwacht wird. Zudem exportiert es grafische Elemente an die Spielwelt damit Spielelemente diese verwenden können.

**Audioausgabe**

stellt die Spielwelt im Bereich Audio dar. Exportiert auditive Elemente an die Spielwelt damit Spielelemente diese verwenden können.

**Eingabe**

Überwacht eingaben welche für die Steuerung des Spielcharakters gedacht sind. Übernimmt dafür Kontrolle über Elemente der Spielwelt.

**Server**

Übernimmt die gesamte Server-Client Kommunikation. Zudem überwacht es verschiedene Elemente welche mit dem Server synchronisiert werden – wie die Spielwelt und der Spielcharakter. Zudem liefert es die Möglichkeit diese vom Server abzurufen und den Benutzer zu verifizieren.