Architektur

Inhalt

[Allgemeine Überlegungen 1](#_Toc400981561)

[libGDX 2](#_Toc400981562)

[Pakete 3](#_Toc400981563)

[UI 3](#_Toc400981564)

[UI::libGDX 3](#_Toc400981565)

[UI::Renderer 3](#_Toc400981566)

[Domain 4](#_Toc400981567)

[Domain::MainController 4](#_Toc400981568)

[Domain::MenuController 4](#_Toc400981569)

[Domain::GameControllers 4](#_Toc400981570)

[Domain::ObjectControllers 4](#_Toc400981571)

[Technical Services 4](#_Toc400981572)

[TechnicalServices::AndroidSDK 4](#_Toc400981573)

[TechnicalServices::Persistence 5](#_Toc400981574)

# Allgemeine Überlegungen

Im Zentrum unserer Architektur-Überlegungen stand zuerst die Frage, ob und wie wir die grafische Repräsentation unserer Spielobjekte von deren Logik und Zustand trennen. Das führte uns zum klassischen Model-View-Controller-Pattern. Nach Absprache mit unserem Dozenten kamen wir aber zum Schluss, dass dies nicht dem angestrebten Domain-Driven Design entsprach, sondern zu einer unnötig künstlichen Trennung der Domänenobjekte führte.

Die Architektur richtet sich nun stark nach dem Domänenmodell und führt Spielobjekte ein, die sowohl einen inneren Zustand haben als auch Logik bereitstellen. Bewusst getrennt werden allerdings die Renderer, mit dem Ziel genug Abstraktion zu erreichen, damit die visuelle Darstellung der Spielobjekte leicht ausgetauscht werden kann. Ausserdem werden die Renderer selber ein gewisses Mass an Logik beinhalten müssen (z.B. Handling von Animationen, Zusammensetzen von Tilesets), die aber nicht direkt mit der Domäne zusammenhängt.

## libGDX

Für die Spieleentwicklung auf Android suchten wir ein Einsteigerfreundliches, aber effizientes Spieleentwicklungsframework. Nach der Analyse von vielen Optionen fiel unsere Wahl auf das libGDX (<http://libgdx.badlogicgames.com/>) Framework. Die Gründe für diese Entscheidung sind in erster Linie folgende:

* Open Source
  + libGDX steht unter der Apache 2.0 Lizenz
* Viele, modulare Features
  + Bietet alles, was die Entwicklung eines Spies unseres Scopes erfordern (und mehr)
  + APIs zu Bibliotheken wie z.B. Box2D für 2D-Physikberechnung sind optional
* In aktiver Entwicklung
  + Version 1.4.1 ist erst kürzlich (10.10.14) erschienen
* Grosse, aktive Community
  + libGDX hat sein eigenes Forum, einen IRC Channel.
  + Auch z.B. auf StackOverflow und Reddit finden sich viele libGDX-Entwickler
* Sehr gut dokumentiert
  + Vollständige Javadoc
  + Diverse Tutorials und Erläuterungen
  + Beispielcode und Demos
* Cross Platform
  + Ermöglicht neben Android auch die Entwicklung für Windows, Mac, Linux, iOS, BlackBerry und HTML5

# Pakete

Im Folgenden werden die Pakete (vgl. Paketdiagramm) mit ihren Aufgaben und Beziehungen beschrieben.

## UI

Das UI-Paket enthält Pakete, die für die Benutzereingaben und die grafische Darstellung verantwortlich sind.

### UI::Renderer

Das Renderer-Paket enthält Objekte (Klassen, evtl. weitere Pakete), die für die grafische Darstellung von Objekten der Spielwelt zuständig sind. So muss beispielsweise der ShipRenderer die Breite des Ship-Objekts auslesen und ist dann dafür verantwortlich, die korrekten Bilddateien zu laden und daraus das Schiff korrekt zusammenzustellen. Der technische Vorteil dieser Trennung ist, dass die grafische Repräsentation von Objekten getrennt ist und so z.B. erst zur Laufzeit bestimmt werden kann, wie ein Objekt gerendert wird (eine Art Dependency Injection). Fachlich macht diese Trennung ebenfalls Sinn, da die Darstellungslogik nichts mit der Domänenlogik zu tun hat.

Die Klassen des UI::Renderer-Pakets nutzen die Grafik-APIs von libGDX.

Die Klassen des UI::Renderer-Pakets greifen auf Objekte des Pakets Domain::GameObjects zu.

### UI::Menus

Das Menus-Paket enthält die Logik für die Menüs. Darunter fällt zum Beispiel das Hauptmenü oder das Einstellungs-Menü.

Die Klassen des UI::Menus-Pakets nutzen die Klasse(n) aus dem UI::Renderer Paket für die visuelle Darstellung.

Die Klassen des UI::Menus-Pakets starten Objekte aus dem Domain::Game Paket.

## Domain

Das UI-Paket enthält die Domänenlogik des Projekts. Sein Aufbau richtet sich stark nach dem Domänenmodell.

### Domain::Game

Die Objekte im Game-Paket sind für die übergreifende Steuerung der Spiellogik zuständig. Jeder Spielmodus wird von einer eigenen Klasse (oder von einem eigenen Paket, falls mehrere Klassen pro Spielmodus anfallen) definiert. Gemeinsame Operationen werden durch Vererbung generalisiert.

Die Objekte im Domain::Game-Paket greifen auf die Config- und Stats Klassen aus dem Domain::User-Paket zu.

Die Objekte im Domain::Game-Paket arbeiten mit den Klassen aus dem Domain::GameObjects-Paket.

Die Objekte des Domain::Game-Pakets nutzen das Input-Handling von libGDX.

Das CareerGame-Objekt des Domain::Game-Pakets greift zusätzlich auf das Domain::Level-Paket zu.

### Domain::GameObjects

### Domain::User

### Domain::Level

## TechnicalServices

### TechnicalServices::Persistence

### TechnicalServices::Android

### TechnicalServices::libGDX

LibGDX ist ein externes Framework (siehe libGDX unter Allgemeine Überlegungen), welches hauptsächlich für das Inputhandling als auch für die Grafikschnittstelle verwendet wird.