Lehrstuhl für Softwaretechnik und Programmiersprachen
Jens Bendisposto
Philipp Körner

**Funktionale Prorammierung – WS 16/17**
**Projektaufgabe**
**In der Version vom 22. Dezember 2016**

# 1 Formales

## 1.1 Abgabe

Die Projektaufgabe muss bis zum 01.03.2017, 23:59 Uhr abgegeben werden. Bitte legen Sie sich dazu einen Account bei `https://bitbucket.org/` an. Erstellen Sie dort ein **privates** git Repository und teilen Sie den Zugriff mit uns. Unsere Usernamen sind `bendisposto` bzw. `koernerp`. Legen Sie das Repository schnellstmöglich an um sicherzustellen, dass es keine Probleme gibt. Es ist uns wichtig, dass Sie regelmässig auf Bitbucket pushen. Gewertet wird der letzte vor der Deadline auf Bitbucket gepushte commit.

Eine vollständige Abgabe umfasst folgende Bestandteile.

1. Der Sourcecode der Anwendung
2. Eine Readme Datei, die die Installation beschreibt.
3. Ein Dokument, dass Ihre Implementierung beschreibt.

## 1.2 Regeln

Die Aufgabe ist eigenständig zu lösen. Sie dürfen mit anderen Teilnehmern gerne Ihre Lösungsansätze diskutieren, müssen Ihre Abgabe aber selber implementieren. Da es sich um Teil der Prüfung handelt, wird die Gesamtprüfung bei einem Täuschungsversuch als nicht bestanden gewertet.

## 1.3 Bewertung

Die Aufgabe wird mit 25 Punkten bewertet. Kriterium für die Bewertung ist in erster Linie die Qualität des Codes. Entwickeln Sie Ihre Anwendung nach den in der Vorlesung diskutierten Prinzipien für Simplicity.

# 2 Aufgabenstellung

Es soll ein Klon für das rundenbasierte Spiel „Konquest“[1] erstellt werden. Im Folgenden sind die Spielregeln beschrieben, die implementiert werden sollen. Diese weichen teilweise von dem Original ab.

## 2.1 Spielregeln

Das Spielfeld ist ein zweidimensionales Raster fester Größe, die bei Spielstart festgelegt wird. Initial werden auf dieses Raster pro Spieler ein Planet sowie eine bei Spielstart angegebene Anzahl neutrale Planeten zufällig verteilt. Auf einem Feld im Raster kann höchstens ein Planet sein.

In einem Zug darf ein Spieler dessen vorhanden Schiffe bewegen. Dabei sind nur Anweisungen erlaubt, die Schiffe von einem vom Spieler kontrollierten Planeten zu irgendeinem anderen Planeten schicken. Anweisungen an sich bewegende Schiffe können nicht mehr geändert werden.

Eine Runde besteht aus je einem Zug jeden Spielers. Am Ende jeder Runde werden auf allen Planeten Schiffe generiert. Dabei soll die Anzahl jedes Mal leicht verschieden sein. Neutrale Planeten generieren grundsätzlich weniger Schiffe als die von Spielern kontrollierten.

Zusätzlich werden die Schiffe, die unterwegs sind, weiterbewegt. Erreichen Schiffe eines Spielers einen freundlichen Planeten, so werden sie dort stationiert. Erreichen sie einen neutralen Planeten oder einen Planeten unter der Kontrolle eines anderen Spielers, so bekämpfen sich die Schiffe.

Das Kampfsystem soll wie folgt ablaufen: für jeden angegriffenen Planeten werden alle ankommenden und bereits stationierten Schiffe betrachtet. Dies bedeutet insbesondere, dass ein Kampf zwischen mehr als zwei Parteien ausgetragen werden kann, wenn beispielweise zwei Flotten verschiedener Spieler gleichzeitig an einem neutralen Planeten ankommen.

In einem Kampf können Schiffe zerstört werden. Grundsätzlich soll das von Ihnen implementierte System folgende Eigenschaften haben:

- Am zu verteidigenden Planeten stationierte Schiffe bekommen einen Bonus auf ihre Kampfstärke.
- Die Partei mit der höchsten Gesamtstärke übernimmt die Kontrolle über den Planeten. Bei einem Unentschieden behält der Verteidiger die Kontrolle.
- Es werden keine Schiffe durch einen Kampf generiert.
- Bei einem sehr ausgeglichenen Kampf (ähnlicher Kampfstärke) soll ein Großteil der am Kampf beteiligten Schiffe des Gewinners zerstört werden.
- Bei einem sehr eindeutigen Kampf (eine große gegen eine sehr geringe Kampfstärke) soll fast kein Schiff des Angreifers zerstört werden.
- Alle am Kampf beteiligten Schiffe von denjenigen Parteien, die nicht gewinnen, werden zerstört.

[1] `https://games.kde.org/game.php?game=konquest`

Das Spiel ist gewonnen, wenn alle steuerbaren Schiffe und alle von aktiven Spielern kontrollierten Planeten einem Spieler gehören.

## 2.2 User Interface

Es soll zwei Benutzerschnittstellen geben: die REPL und eine graphische Schnittstelle. In der REPL soll ihr Namespace mit use geladen werden und dann per Hand gespielt werden.

In beiden Interfaces sollen nur Zugehörigkeiten von Planeten und Spielern sowie Informationen des aktiven Spielers (Anzahl stationierter Schiffe und Flottenbewegungen) sichtbar sein. Wir gehen davon aus, dass die Spieler sportlich sind und während gegnerischer Züge nicht auf den Monitor schauen, nach Beenden des Zuges nicht auf den vorhergenden Inhalt der REPL schauen und nicht den internen Spielzustand selbst abfragen.

Folgende Funktionen sollen auf der REPL verfügbar sein:

- `(init! ...)` startet ein Spiel mit vorgegebener Größe des Spielfeldes sowie Anzahl neutraler Planeten. Wie die Argumente hier genau aussehen sollen, ist Ihnen überlassen.
- `(whose-turn)` gibt aus, welches der derzeit aktive Spieler ist.
- `(show-board!)` gibt eine sinnvolle, menschenlesbare Repräsentation des Spielfeldes aus.
- `(send! from to amount)` gibt die Anweisung, von einem Planeten „from“ zum Planeten „to“ `amount` viele Schiffe zu schicken. Sollte dies dem Spieler nicht erlaubt sein, soll eine entsprechende Fehlermeldung ausgegeben werden, aus welcher der Grund ersichtlich ist. Um die Planeten zu spezifizieren, sollen Sie sich selbst ein sinnvolles Format ausdenken.
- `(movements!)` soll alle aktuellen Flottenbewegungen des aktuellen Spieler ausgeben, die Start- und Zielplanet sowie Anzahl der Schiffe und die Zeit zur Ankunft in Runden beinhaltet.
- `(end-turn!)` beendet den aktuellen Zug des aktiven Spielers und gibt die Kontrolle an den nächsten Spieler weiter.
- `(distance from to)` ist eine Hilfsfunktion für den Spieler, um die Distanz zwischen zwei Planeten zu messen. Die Rück- oder Ausgabe soll die benötigte Zeit in Runden sein, die eine Flotte benötigt, um vom Planeten „from“ aus den Planeten „to“ zu erreichen.
- `(save-to! path)` und `(load-from! path)` sollen das derzeitige Spiel speichern bzw. laden. „path“ soll für die dafür zu verwendende Datei stehen.

Bei dem graphischen User-Interface ist Ihre Kreativität gefragt. Sie können die Applikation als Standaloneanwendung (z.B. mit Swing, SWT, JavaFX, etc.) schreiben oder auch als Webanwendung. Die GUI soll die Ausführung der wesentlichen Funktionen (Starten/Laden/Speichern und Spielen) erlauben. Wenn Sie keinen Zugriff auf das Dateisystem haben (zum Beispiel, weil Sie ihre Anwendung als reine ClojureScript Anwendung schreiben) ist es auch akzeptabel, wenn man den Dateiinhalt in ein Textfeld kopieren muss.

### 2.3 Erlaubte Bibliotheken

Jede Bibliothek, die sie wollen, außer Bibliotheken, die selbst „Konquest“ implementieren.

### 2.4 Implementierung

Ihre Anwendung soll ein funktionales Programm in Clojure oder ClojureScript sein. Wenn es **erforderlich** ist, können kleine Teile in anderen Sprachen implementiert werden. Bibliotheken, die Sie verwenden, dürfen auch in anderen Sprachen implementiert sein. Es müssen aber Bibliotheken sein, die per öffentlichem Maven Repository verfügbar sind und es sollen keine nativen Bibliotheken benutzt werden[2].

### 2.5 Docstrings, Tests, Schemas, Types, Contracts

Bitte schreiben sie für alle öffentlichen Funktion einen Docstring, der Funktion und Parameter **kurz und präzise** beschreibt. Ob Sie Tests, Typen, Contracts oder Schemas verwenden bleibt Ihnen überlassen. Es wird aber dazu geraten die Codequalität durch eines oder mehrere dieser Werkzeuge sicherzustellen. Im Zweifel wird das Vorhandensein von *sinnvollen* Tests, etc. zu Ihren Gunsten gewertet.

## 3 Dokumentation

Genauso wichtig wie die Anwendung selber ist die Dokumentation des Projekts. Sie sollen hier nicht beschreiben, was im Quelltext sehr einfach nachgelesen werden kann. Das Dokument soll die Architekturentscheidungen dokumentieren. Insbesondere sollen Sie beschreiben, wie Simplicity erreicht wurde, wo es Tradeoffs gab, und wo komplexe Konstrukte nötig waren.

Es ist wichtig, dass Sie über Ihren Quellcode reflektieren und verschiedene Ansätze ausprobieren. Dokumentieren Sie auch Ansätze, die Sie verworfen oder radikal geändert haben. Wir empfehlen Ihnen während der Implementierung eine Art Forschungstagebuch zu führen, in dem Sie ihre Ansätze und Ideen festhalten. Das wird die Erstellung des Dokuments radikal vereinfachen.

## 4 Installationsanleitung

Die Installationsanleitung soll **knapp** beschreiben, welche Schritte zum Bauen und Starten der Anwendung nötig sind. Benutzen Sie nach Möglichkeit das Buildtool leiningen.

## 5 Fragen

Bei Fragen zur Aufgabenstellung wenden Sie sich bitte an Philipp Körner: p.koerner@hhu.de.

[2] Ihr Programm muss auf Jens' Mac Book und Philipps Notebook laufen