

Durak

Software Engineering



Stephan Perren, Felix Mayer

Gliederung

Version Control System

Build System

MVC Architektur

Continuous Deployment

Design Patterns

Components and Interfaces

Dependency Injection

File IO

Docker

TUI

GUI

Version Control System

► Git:

- Verbessert das Arbeiten im Team.
- Sehr komfortabel zu bedienen.
- Es können umfangreiche **Statistiken** auf **GitHub** eingesehen werden.
- Einbindung in **IntelliJ** möglich.
- Ermöglicht das Entwickeln auf verschiedenen **Branches**.
- Ermöglicht komfortable Anbindung von **TravisCI** bzw. **Coveralls**.
- Alte Revisionen gehen **nicht** verloren und können problemlos erneut "ausgecheckt" werden.

Version Control System

► Contributors:

Jun 7, 2015 – Jul 4, 2019

Contributions: **Commits** ▾

Contributions to master, excluding merge commits



Build System

► **sbt:**

- Open Source Build-Tool für Scala- bzw. Java-Projekte.

```
import sbt.CrossVersion

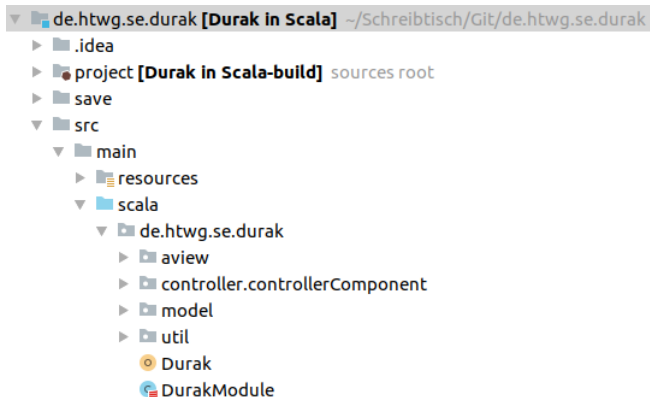
name      := "Durak in Scala"
organization := "htwg"
version   := "0.0.1"
scalaVersion := "2.12.7"

scalacOptions ++= Seq(
  "-encoding", "utf8", // Option and arguments on same line
  "-Xfatal-warnings", // New lines for each options
  "-deprecation",
  "-unchecked",
  "-Xcheckinit",
  "-language:implicitConversions",
  "-language:higherKinds",
  "-language:existentials",
  "-language:postfixOps",
)

// ...
```

MVC Architektur

► Model-View-Controller Architektur:



Continuous Deployment

► TravisCI:

- Baut bei jedem "push" automatisch das Projekt und führt die Tests aus.
- Der Build-Status des Projekts kann in GitHub eingesehen werden.



▶ TravisCI:



✓ master Update README.md

 Compare a76d810...a8cb8ab 

 Stephan Perren

 Ran for 2 min 52 sec
 17 days ago

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ▶ ↺ 🔍 ↻

Continuous Deployment

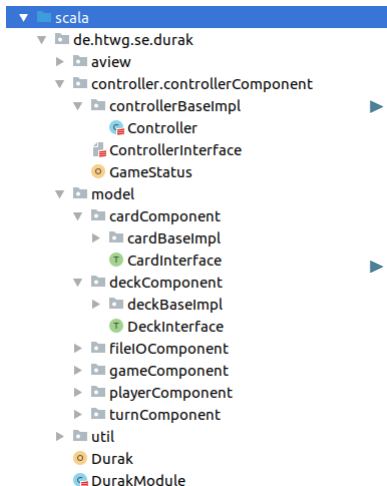
► Coveralls:



Design Patterns

- ▶ Observer Pattern
- ▶ Memento Pattern
- ▶ Dependency Injection Pattern
- ▶ Singleton Pattern

Components and Interfaces



- ▶ Interaktion der Komponenten erfolgt über **Interfaces**.
- ▶ Vereinfacht es Komponenten **auszutauschen**.

Dependency Injection

- ▶ **Google Guice:**
 - ▶ Leichtgewichtiges Dependency Injection Framework
- ▶ **Scala Guice:**
 - ▶ Scala Wrapper für Google Guice.

Dependency Injection

► Scala Guice:

DurakModule.scala:

```
package de.htwg.se.durak

import com.google.inject.AbstractModule
import de.htwg.se.durak.controller.controllerComponent.ControllerInterface
import de.htwg.se.durak.controller.controllerComponent.controllerBaseImpl
import de.htwg.se.durak.model.fileIOComponent.FileIOInterface
import de.htwg.se.durak.model.fileIOComponent.fileIOJsonImpl.FileIO
import de.htwg.se.durak.model.gameComponent.GameInterface
import de.htwg.se.durak.model.gameComponent.gameBaseImpl.Game
import net.codingwell.scalaguiceScalaModule

class DurakModule extends AbstractModule with ScalaModule {
  def configure(): Unit = {
    bind[GameInterface].to[Game]
    bind[ControllerInterface].to[controllerBaseImpl.Controller]
    bind[FileIOInterface].to[FileIO]
  }
}
```

Dependency Injection

► Scala Guice:

Durak.scala:

```
package de.htwg.se.durak

import com.google.inject.{Guice, Injector}
import de.htwg.se.durak.controller.controllerComponent.ControllerInterface
import de.htwg.se.durak.aview.Tui
import de.htwg.se.durak.aview.gui.Gui

import scala.io.StdIn._

object Durak {
  val injector: Injector = Guice.createInjector(new DurakModule)
  val controller: ControllerInterface = injector.getInstance(classOf[ControllerInterface])
  val tui: Tui = new Tui(controller)
  val gui: Gui = Gui(controller)

  def main(args: Array[String]): Unit = {
    // ...
  }
}
```

File IO

- ▶ Es ist möglich die Spielstände im **XML-Format** bzw. **JSON-Format** abzuspeichern.
- ▶ Die abgespeicherten Spielstände lassen sich auch wieder laden und man kann den abgespeicherten Spielstand fortsetzen.

Docker

- Unser Durak-Spiel lässt sich auch in einem Dockercontainer ausführen.



gruselopi/de.htwg.se.durak ☆

By [gruselopi](#) • Updated 17 days ago

Durak in scala; repo for our software project

Container

TUI

```
A new player was added  
A new player was added  
A new game was created.
```

```
=====
```

```
players turn: Herz Drei,Herz Vier,Herz Fünf,Herz Sechs,Kreuz Ass  
Trump: [Karo] Drei  
Attacker: Felix  
Defender: Stephan  
Neighbor: Felix  
Cards to block:  
Blocked Cards:  
cards: Herz Drei,Herz Vier,Herz Fünf,Herz Sechs,Kreuz Ass  
winners: List()
```

```
=====
```

```
A card was layed.
```

GUI

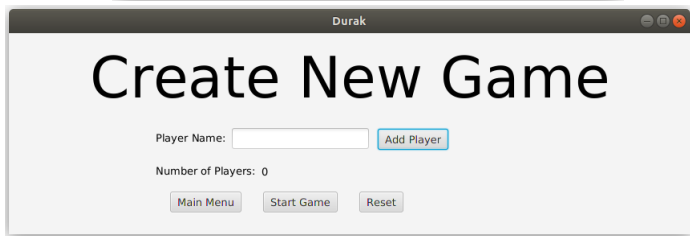
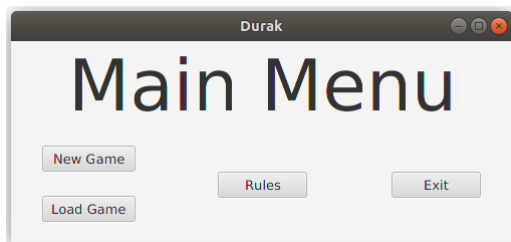
- ▶ **ScalaFX mit FXML:**

- ▶ Scala Wrapper für JavaFX.

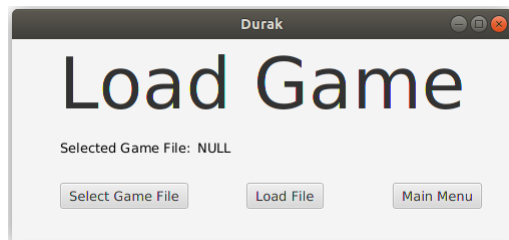
- ▶ **Scenebuilder:**

- ▶ Das User-Interface kann mittels Drag-and-Drop erstellt werden.
 - ▶ Sehr komfortabel zu bedienen.
 - ▶ Unterstützt JavaFX bzw. ScalaFX.

GUI



GUI



Durak

Spielregeln

Vorbereitung:

Anzahl Karten im Deck: 52
Maximale Anzahl Spieler: 10
Zu Beginn des Spiels erhält jeder Spieler 5 Handkarten.

Spielverlauf:

Ein zufällig ausgewählter Spieler darf beginnen.
Es wird rundenweise im Uhrzeigersinn gespielt. Eine Runde beginnt mit dem Angriff des Angreifers, welcher den Spieler seiner linken (den Verteidiger) angreifen darf. Der Verteidiger muss nun die vom Angreifer gelegten Karten decken.
Gelingt ihm dies, dürfen beide Nachbarn des Verteidigers weitere Karten hinzuwerfen. Es können jedoch nur Karten dazugeworfen werden, welche die selbe Wertigkeit besitzen, wie eine der bereits auf dem Spielfeld liegenden Karten. Die dazugeworfenen Karten muss der Verteidiger nun ebenfalls decken (sobald dieser an der Reihe ist), wodurch der Zyklus wieder von vorne beginnt. Dies wird solange Wiederholt, bis beide Nachbarn des Verteidigers zufrieden sind, also keine Karten mehr dazu werfen möchten.
Gelingt es dem Verteidiger nicht, den Angriff des Angreifers abzuwehren, so muss dieser die Karten auf dem Tisch aufnehmen und der Spieler zu seiner linken ist nun an der Reihe anzugreifen.
Sobald ein Spieler einen Zug an einen anderen Spieler übergibt muss dieser Karten aufziehen, falls er unter 5 Karten auf der Hand hat. Es werden so viele Karten gezogen, bis der Spieler wieder 5 Handkarten besitzt.
Der Spieler welcher als erstes keine Handkarten mehr besitzt, gewinnt das Spiel.
Das Spiel wird solange weitergespielt bis es nur noch einen Spieler gibt, der als einziges noch Handkarten besitzt. Derjenige ist dann der Verlierer.

Karten:

Die Spielkarten geordnet von der niedrigsten Wertigkeit zur höchsten Wertigkeit:

2, 3, 4, 5, 6, 7, 8, 9, Bube, Dame, König, 10, Ass

Trumpfkarte:

Zu Beginn des Spiels wird eine Trumpfkarte festgelegt (die letzte Karte des Stapels). Die Farbe dieser Karte legt die Trumpffarbe fest, d.h. alle Karten mit dieser Farbe sind nun Trumpfkarten. Diese Karte ist für alle Spielteilnehmer sichtbar und kann auch aufgezogen werden.
Eine Trumpfkarte kann jede normale Karte decken. Eine Trumpfkarte kann mit einer anderen Trumpfkarte mit höherer Wertigkeit gedeckt werden.

Angriff:

Ein Angreifer darf beliebig viele Karten der gleichen Wertigkeit legen um den Verteidiger anzugreifen.

Verteidigen:

Ein Verteidiger muss alle ungedeckten Karten auf dem Spielfeld decken, und zwar solange bis beide Nachbarn zufrieden sind und nichts mehr dazuwerfen möchten. Eine auf dem Spielfeld liegende Karte kann nur mit der selben Farbe und einer höheren Wertigkeit gedeckt werden. Ausnahme Trumpfkarten.
Ein Verteidiger hat auch die Möglichkeit einen Angriff weiterzuschieben, indem er eine Karte einer anderen Farbe mit der selben Wertigkeit wie die auf dem Spielfeld liegenden Karten besitzt. Dadurch greift der Verteidiger nun den Spieler zu seiner linken an. Dies ist jedoch nur möglich, solange der Verteidiger noch keine der Angriffskarten gedeckt hat.

Main Menu

GUI

