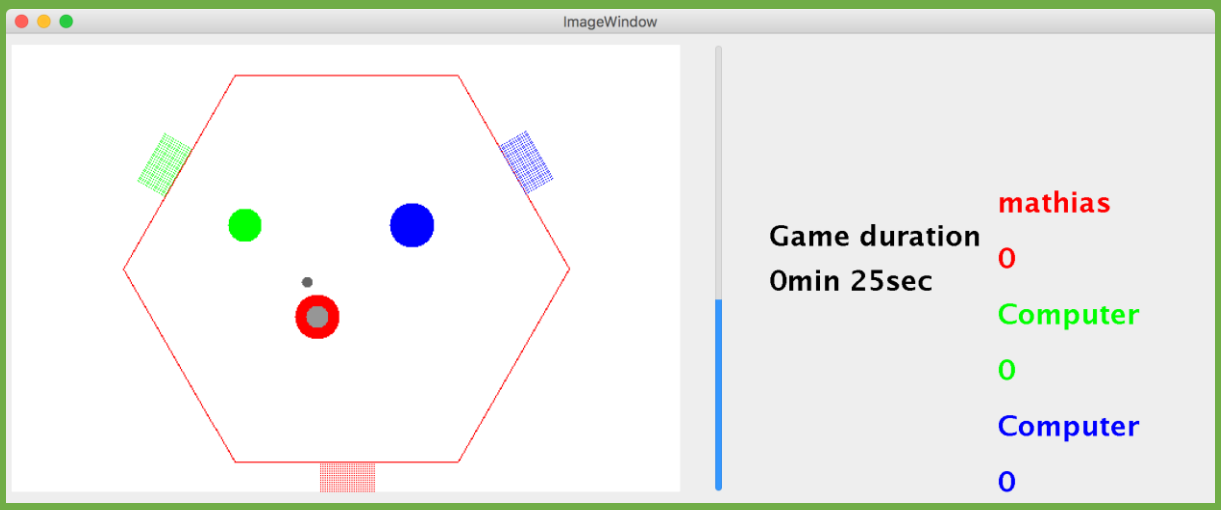
Andrea Tuccillo
Mathias Schmid
Maximilian Reber
Elias Arnold

Handbuch zum Spiel

Hexagon



1. Installation des Spiels

a) Erstellen einer Jar-Datei

Um unsere Software zu installieren, wird das mitgelieferte Buildfile „build.xml“ ausgeführt. Dieses Buildfile wird mit Hilfe des Tools Ant der Apache Software Foundation gestartet. Ant wandelt die geschriebenen Quelltexte in ein Computerprogramm um. Falls Sie das Open Source Tool Apache Ant noch nicht auf ihrem Computer installiert haben, können Sie es von der Webseite: http://ant.apache.org/srcdownload.cgi downloaden. Die Installation ist etwas schwierig, ist aber im File install.html, das ebenfalls unter http://ant.apache.org/manualdownload.cgi heruntergeladen werden kann, bestens beschrieben.

Apache Ant bietet leider keine Graphische Benutzeroberfläche. Es kann ausschliesslich über die Kommandozeile gestartet werden. Wechseln Sie (mit den `cd` Befehl in der Kommandozeile) also in den entsprechenden Ordner, indem das Buildfile liegt, und starten Sie mit `ant`(Befehl in der Kommandozeile) die Erstellung einer ausführbaren Jar-Datei.

b) Starten des JUnit Tests

Mit Hilfe derselben Software, Apache Ant, kann Hexagon auch auf „Herz und Nieren“ getestet werden. „Auf Herz und Nieren“ ist an dieser Stelle aber etwas übertrieben, da nur eine einzige Klasse (Ein Javaprojekt ist aus verschiedenen Klassen aufgebaut, und enden mit „.java“) getestet wird. Geben Sie auf der Kommanozeile den Befehl: `ant junit` ein, und bestätigen Sie mit betätigen der Leertaste. Das System führt anschliessend ein Unit Test mit der Klasse Lobby.java durch. Die zu testende Klasse ist wiederum aus Methoden aufgebaut (jede Methode versorgt die Klasse, der sie angehört, mit spezifischen Funktionalitäten). Sie sollten nun als Ausgabe eine Auflistung der einzelnen Testfälle erhalten. Falls der Test fehlschlägt, wird unter der entsprechenden Zeile die Fehlermeldung angezeigt. Wurde die zu testende Klasse aber sauber Programmiert und falsche und unerwünschte Ausgaben abgefangen, verlaufen alle Tests erfolgreich, und zwei Zeilen unter den letzten Test wird `BUILD SUCCESSFUL` angezeigt.

Falls Sie sehen möchten, welche falschen Eingaben das Programm genau abgefangen hat, scrollen Sie im Konsolenfenster ein Stück nach oben. Für jeden erzwungenen Fehler, der abgefangen wurde, wird eine Instanz einer Exception erzeugt, die eine Fehlermeldung beinhaltet. Eine Klasse (hier Exception) kann man sich wie ein Bauplan eines Hauses vorstellen. Eine Instanz einer Klasse wäre dann ein Haus, das genau nach diesem Bauplan gebaut wurde. Wie bereits angetönt, beinhaltet jede Instanz der Exception Klasse eine individuelle Fehlermeldung. Diese Meldungen werden unter `Standard Output` ausgegeben.

2. Starten eines Spiels

a) Java installieren

Um eine erstellte Jar-Datei auszuführen, wird eine aktuelle Version von Java benötigt. Letztere kann unter http://www.java.com/de/download/win10.jsp gratis bezogen werden.

Unter Windows: Fügen Sie Ihrer Umgebunsvariable Path zwei neue Pfade hinzu

b) Ein Server starten

Man wechselt wiederum (mit der Konsole) in das Verzeichnis, in den die im zweiten Kapitel erstellte Jar-Datei liegt. In dieser Jar-Datei sind alle von uns geschriebenen Klassen zusammengefasst. Um einen Server zu starten, führt man den Befehl: `java –jar Hexagon.jar server <Port>` aus. Folgendes kleine Fenster sollte sich nun auf Ihrem Bildschirm öffnen:

Abb. 1: Das Serverfenster

Schliessen Sie das Fenster nicht! Trotz seiner geringen Grösse ist es für die Funktionalität des Spiels essentiell. Der Server berechnet einerseits Spieler-/Banden- und Ballpositionen sowie deren Kollisionen, andererseits ermöglicht er auch die Kommunikation mit anderen Mitspielern in Ihrem Netzwerk.

Dem letzten Befehl wurden drei Kommandozeilenargumente mitgegeben:

1. Hexagon.jar - Der Name der Jar-Datei, die ausgeführt werden soll
2. server - Was soll gestartet werden? Ein Client oder der Server?
3. <Port> - der Port, der für die Kommunikation verwendet werden soll

Das letzte Argument, die Portnummer, kann aber beliebig verändert werden. Es gibt aber Ports, die standardmässig für den User gesperrt sind (weil sie von einem anderen Programm/Protokoll verwendet werden). Und auch solche die von nachträglich installierter Software beansprucht werden. Je nach System kann es jedoch zu Komplikationen mit Portüberschneidungen kommen. Nachfolgend eine Tabelle mit Ports die standardmässig offen sind. Bevor Sie die Portnummer aber ändern, prüfen Sie bitte auf Ihrem System, ober dieser nicht durch eine andere Anwendung genutzt wird. (Mit dem Befehl „`netstat -a`“ in der Kommandozeile, können Sie sich alle verwendeten Ports anzeigen lassen)

	Standardmässig genutzt	Standardmässig nicht genutzt
Portnummern	0 – 1023	1024 - 65535
Beispiele	HTTP , SMTP, POP, IMAP, … (Offizielle Protokolle)	ArmA, Halo (Computerspiele), TeamSpeak, BitTorrent, …

3. Einem Spiel beitreten

a) Was macht der Server

Ein Server kann symbolisch mit einem Spielcasino verglichen werden. Ein sog. Client kann nun dem Server die Spielzüge des Users zustellen, und bekommt als Antwort die benötigten Daten (neuer Punktestand, Reaktionen der Mitspieler, allfällige Verletzungen der Spielregeln, etc.) zurück.

b) Starten eines Clients

Um einen neuen Client zu starten verfährt man ähnlich wie beim Erstellen eines Servers. Jedoch müssen die Kommandozeilenargumente etwas abgeändert werden. Man wechselt mittels den `cd` (Change Directory) Befehl wieder in den Ordner, indem die Jar-Datei „Hexagon.jar“ liegt. Der nächste Befehl lautet aber: `java -jar Hexagon.jar client localhost:<Port>` Wobei dieselbe Portnummer wie beim Server verwendet werden sollte. Um die Verbindung zu testen, starten Sie am bestem gleich mehrere Clients. Jeder Client sollte nun jeden anderen, existierenden Client kennen. Damit ein Client die Namen der neu sich in der Lobby befindlichen Clients anzeigt, muss das Clientfenster aktualisiert werden. Dies kann man mit einem Click auf den Button „Update Lobby“ bewerkstelligt werden:

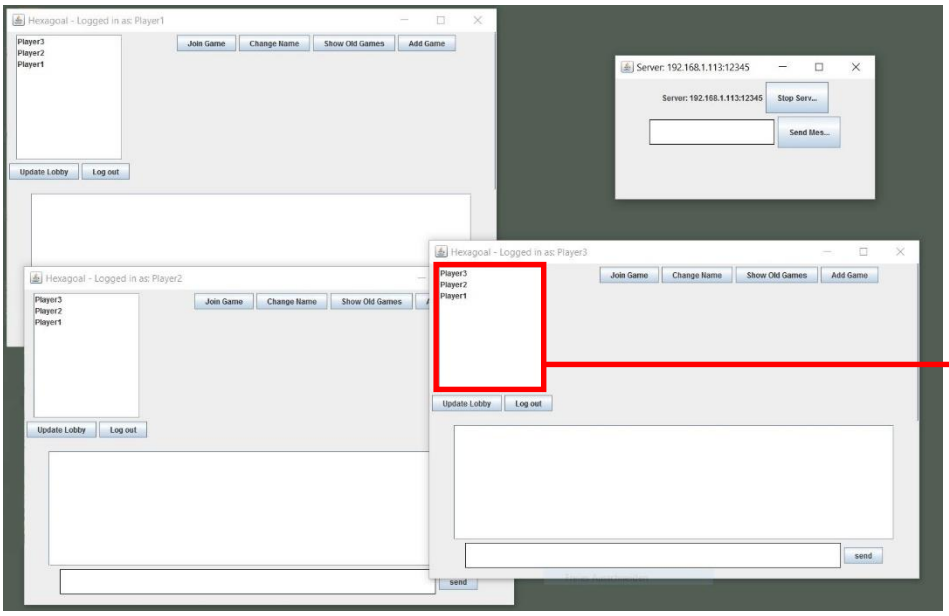


Abb. 2: Drei Clients, die über den gleichen Server (oben links im Bild ist das Serverfenster) kommunizieren. Bei jedem Client wurde „Update Lobby“ durchgeführt.

Mit einem Click auf den Button „Log out“, meldet sich der entsprechende Client korrekt vom Server ab. Sofern die anderen Clients, die immer noch in der Lobby angemeldet sind, ein „Update Lobby“ durchführen, verschwindet der abgemeldete Client aus ihrer Spielerliste.

c) Starten eines Spiels

Um spielen zu können, muss zuerst ein Spiel erstellt werden. Betätigen Sie dazu den Knopf „AddGame“. Alle anderen Client in der gleichen Lobby sehen nun Ihr Spiel, und können diesem mit einem Doppelklick beitreten. Der Spieler, der das Spiel erzeugt hat, wird zum Game Leader ernannt. Nur er kann das Spiel starten. Es können maximal drei Spieler einem Spiel beitreten. Sind es nur zwei oder will man alleine spielen, werden die nicht durch Clients besetzten Spielerplätze durch den Computer dargestellt:

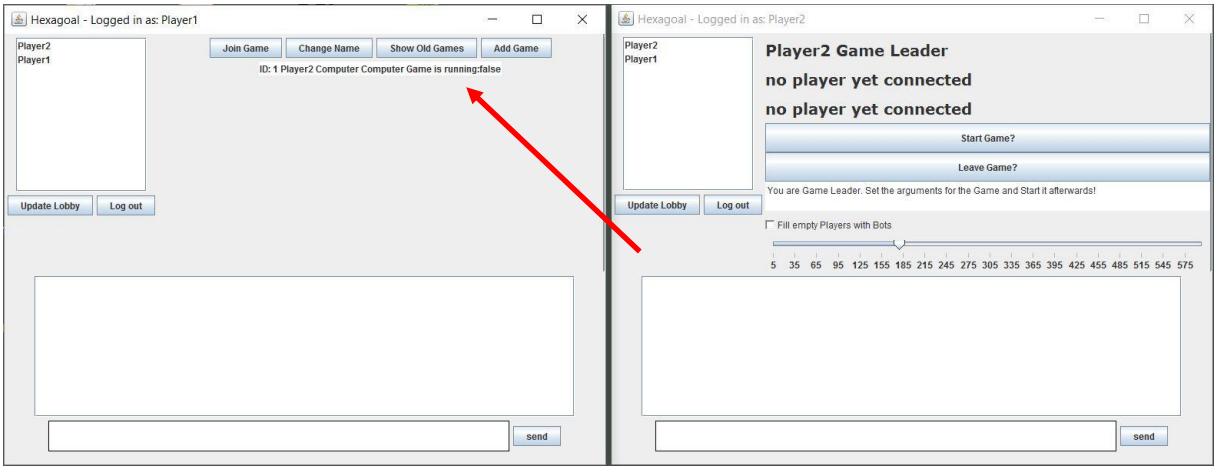


Abb. 3: In diesem Fallbeispiel hat der Client „Player2“ ein Spiel erstellt. Es wird umgehend allen anderen Spielern in der gleichen Lobby angezeigt (roter Pfeil).

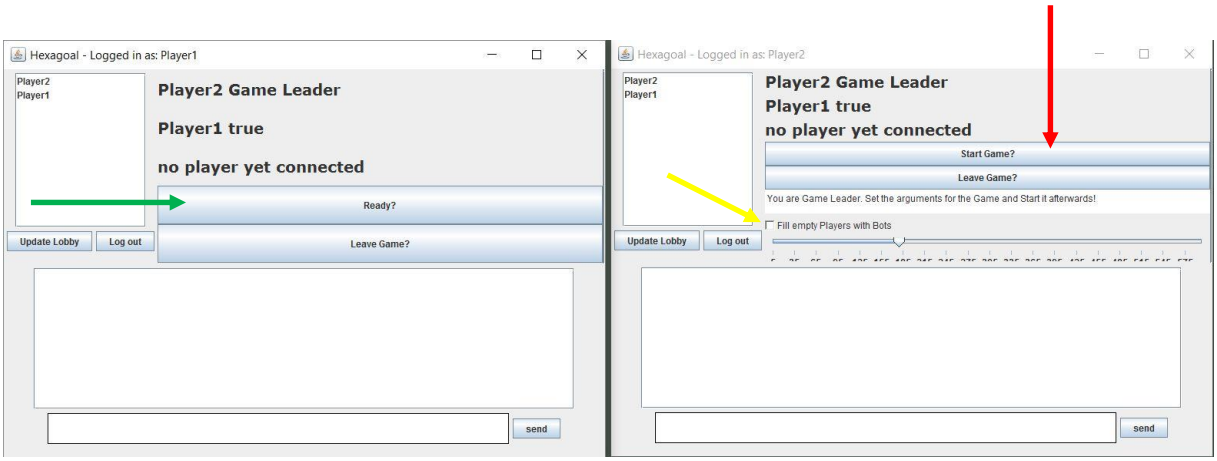


Abb. 4: „Player1“ ist dem Spiel beigetreten. Damit „Player2“ das Spiel starten kann, muss der beigetretene die Schaltfläche „Ready?“ betätigen (grüner Pfeil). Um den dritten, benötigten Spieler durch den Computer darstellen zu lassen, muss „Player2“ die Checkbox „Fill empty Players with Bots“ aktivieren (gelber Pfeil). Nur dann kann das Spiel gestartet werden (roter Pfeil).

Abb. 5: Das Serverfenster mit eingeblendeter IP-Adresse

d) Spielen über das lokale Netzwerk

Fast wie beim Erstellen des Servers dürfen auch hier Kommandozeilenargumente angepasst werden. Auf die Veränderlichkeit der Portnummer-Angabe wurde unter h) Starten eines Clients schon hingewiesen. Es besteht jedoch auch die Möglichkeit, dieses Spiel über das lokale Netzwerk zu spielen. Läuft der Server, auf den Sie zugreifen möchten, nicht auf Ihrem PC, sondern auf einem anderen Rechner in Ihrem Netzwerk, müssen Sie das Kommandozeilenargument vor dem Doppelpunkt mit der IP-Adresse des Zielrechners tauschen. In Ihrer Konsole sieht das dann etwa so aus:

```
java -jar Hexagon.jar client 127.0.0.1:12345
```

Abb. 6: Zugriff auf einen Server in einem Netzwerk der Klasse C.

e) Überprüfen der Verfügbarkeit vom Server/ anderen Clients

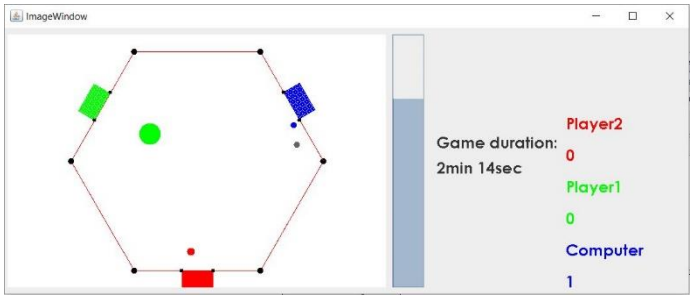
Hat sich ein Client erfolgreich auf einem Server angemeldet, wird mit Hilfe eines Pings überprüft, ob die jeweils andere Seite verfügbar ist. Wenn die Verbindung länger als 60 Sekunden unterbrochen ist, wir der entsprechende Client vom Server aus der Spielerliste entfernt. Ein Client hat also genau eine Minute Zeit, um die Verbindung wieder herzustellen. Danach muss wieder ein neuer Client gestartet werden, um sich erneut mit dem Server zu verbinden.

4. Das Spiel

a) Ziel

Ziel des Spiels ist es, möglichst wenige Punkte zu sammeln. Das tönt vielleicht zuerst etwas komisch, denn es gibt einen gravierenden Unterschied zum herkömmlichen Fussball. Im Spiel Hexagon wird jedes Tor mit einem Minuspunkt bewertet. Dieses ausgefallene Punktesystem verleitet aber, ausschliesslich zu verteidigen und gegnerische Angriffe abzuwehren. Als Gegenmassnahme haben wir uns überlegt, das Angreifen einfacher- und das Verteidigen schwieriger zu machen. Die Umsetzung ist in einer Änderung des Radius des Spielers (der als Kreisscheibe dargestellt wird) geschehen. Je weiter der Spieler von seinem Tor entfernt ist, desto grösser wird er:

Abb. 7: Der von Rechner simulierte Spieler „Computer“ sowie der reale Spieler „Player2“ werden hier viel kleiner dargestellt als der grüne Spieler „Player1“, da sie sich in der Nähe ihres eigenen Tores befinden.



b) Chatfunktion

In der Lobby hat jeder angemeldete Client ausserdem die Möglichkeit, Nachrichten an beliebige andere Clients, oder an die ganze Lobby zu schicken. Um eine Global-Message (Nachricht an die gesamte Lobby) loszuschicken, verfährt man gleich, wie bei jedem bekannteren Chatprogramm. Man tippt im Eingabefeld seine Nachricht ein, und versendet diese anschliessend mit betätigen des „send“ Bttons. Um einen anderen Mitspieler direkt anzusprechen stellt man die Zeichenfolge `/say` vor die Eingabe. (Sie können sich diese Zeichenkette auch generieren lassen, indem Sie den gewünschten Empfängernamen in der Spielerliste auswählen).

Falls Sie schon einem Spiel beigetreten sind, oder selbst eines erstellt haben, dann existiert auch die Möglichkeit das Präfix `/game` zu verwenden. Damit versenden Sie die Nachricht nur an die Spieler/Clients in Ihrem Spiel.

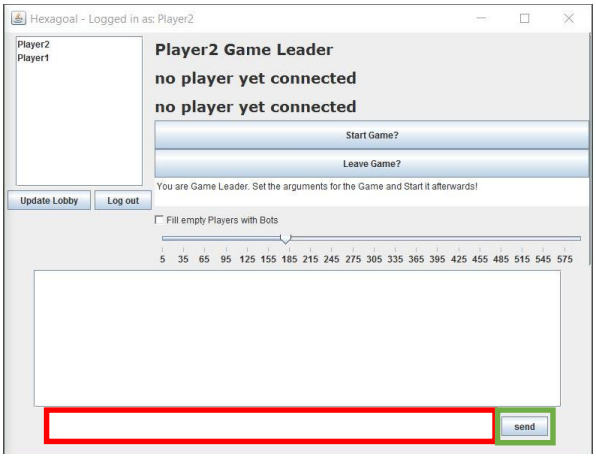


Abb. 8: Lobbyfenster mit Eingabefeld(rot) und „send“-Button(gelb)

c) Chronik

Wenn ein Spiel beendet wird, werden alle relevanten Daten dieses Spiels auf Serverseite gespeichert. Wenn Sie in der Lobby die Schaltfläche „Show Old Games“ betätigen, können Sie eine Liste des Servers einsehen, auf dem Sie angemeldet sind, welche die Ergebnisse aller vergangenen Spiele (nur auf diesem Server) bereithält. Es werden sowohl die Nummer des Spiels, das Spieldatum, die Minuspunkte der drei Spieler, sowie die totale Anzahl an gefallenen Toren angezeigt.

5. Entwicklung

Während der Entwicklung des Spiels, forderten uns zahlreiche Hürden und Hindernisse immer wieder aufs Neue heraus. Obwohl unser Spiel nur wenigen Bedingungen gerecht werden musste, zog sich die Entwicklung einiger Komponenten enorm in die Länge. Zudem stellten wir fest, dass es (zumindest bei uns) unmöglich ist, dass eine Person eine ganze Klasse, die von einer anderen Person geschrieben worden ist, vollständig verstehen kann, und diese ergänzen kann. Soll also eine neue Komponente entwickelt werden, die auf bereits existierenden Klassen aufbaut, muss dies von der gleichen Person gemacht werden, die eben diese zu Grunde liegenden Klassen geschrieben hat. Es sei denn, alle Methoden dieser Klassen sind schon bekannt, diese wurden auf Korrektheit hin überprüft, und müssen auch nicht mehr verändert werden. Eine solche Darstellung aller Methoden nennt man ein „UML Klassendiagramm“:

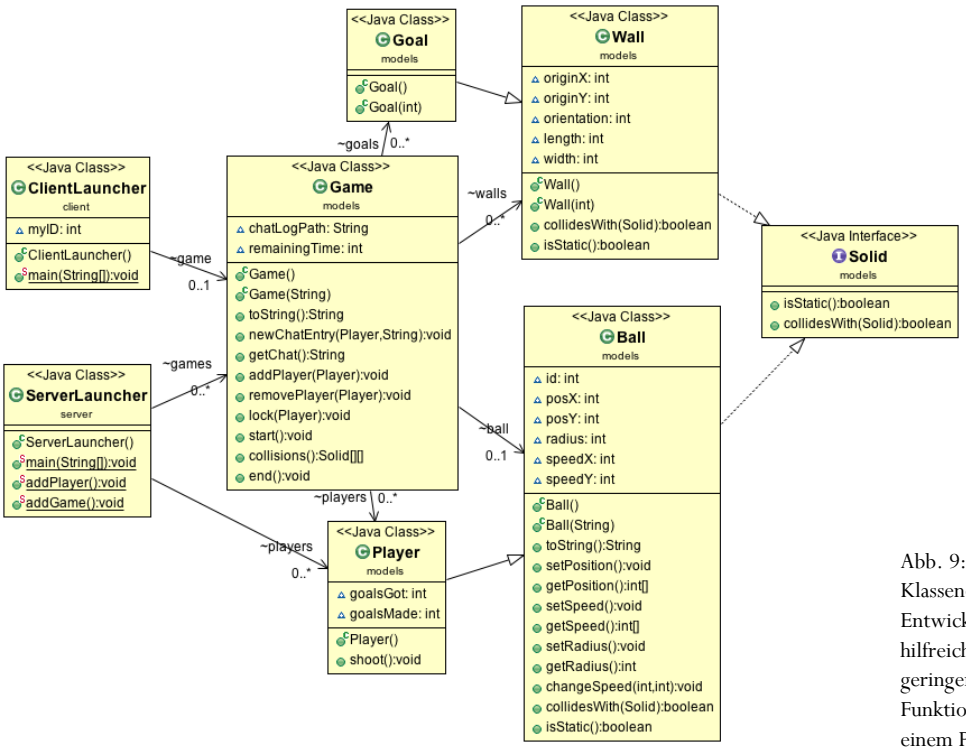
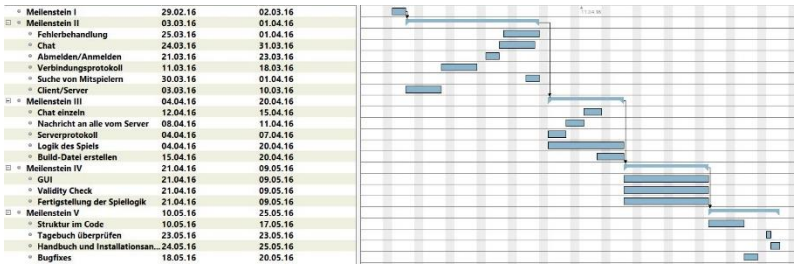


Abb. 9: Eines unserer UML Klassendiagramme. Für den Entwickler ist das eine sehr hilfreiche Grafik, um mit geringem Aufwand in die Funktionsweise jeder Klasse in einem Projekt einzuarbeiten.

Leider fingen wir erst sehr spät mit den Erstellen von UML Klassendiagrammen an, was uns einerseits den Einstieg enorm erschwerte, und andererseits vom Einhalten unseres anfänglich definierten Zeitplanes abhielt:

Abb. 10: Unser Zeitplan, erstellt mit GanttProject©. Die einzelnen Balken/Aufgaben waren an die Mitarbeiter aufgeteilt. Einhalten konnten wir diese Vorgabe leider nicht so stark.



6. Qualitätssicherung

a) Softwarequalitätskonzept

Bei der Implementierung unseres Spiels möchten wir unsere Softwarequalität sicherstellen, indem wir uns am konstruktiven Qualitäts-Konzept „Logging“ orientieren. Sowohl für den Server als auch für den Client haben wir auf das in der Vorlesung vorgeschlagene Tool „self4j“ zurückgegriffen. Nachfolgend ein paar Beispiele aus den Client Klassen:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Abb. 11: Einbindung der beiden benötigten Klassen, um das Logger-Objekt zu erzeugen (GameClient.java)

Abb. 12: Für die Klasse GameClient wird eine Instanz des Logger-Objekts erzeugt

```
log = LoggerFactory.getLogger(GameClient.class);
```

```
} catch (IOException e) {
    log.error(e.getMessage());
}
```

Abb. 13: Das Logger-Objekt erzeugt eine Error-Message, mit der Beschreibung der Exception (Natürlich nur wenn der catch-Block durchlaufen und eine Exception geworfen wird)

b) FMC-Modell (Qualitäts-Merkmale/-Teilmerkmale)

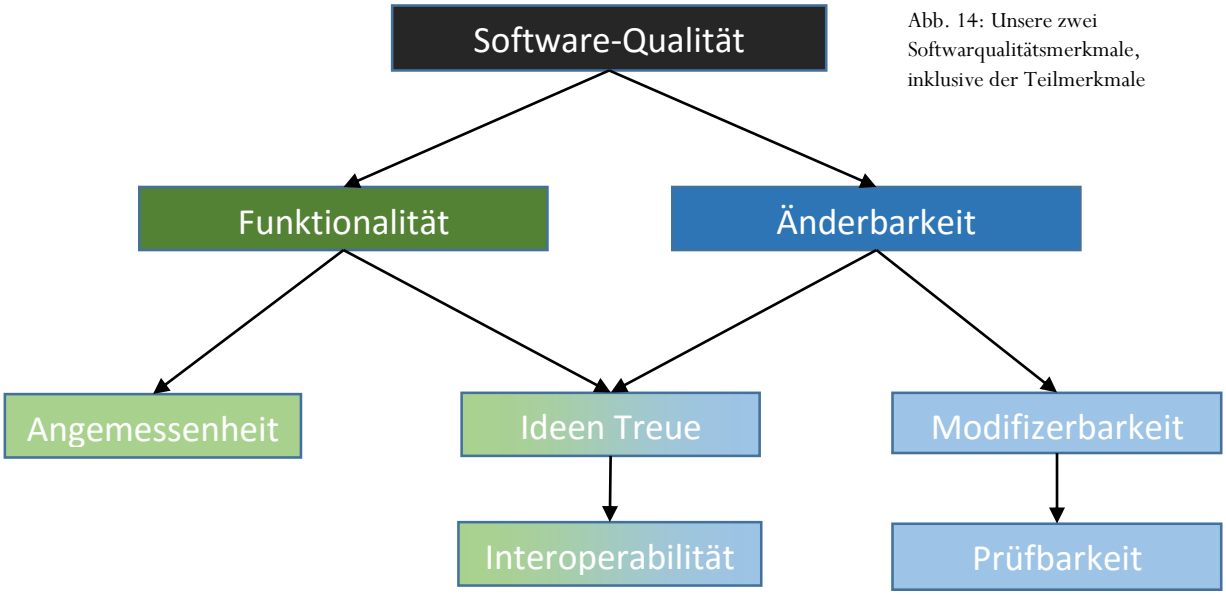


Abb. 14: Unsere zwei Softwarqualitätsmerkmale, inklusive der Teilmerkmale

c) Zu den Qualitätsmerkmalen (Indikatoren)

Funktionalität
Am Ende unseres Projektes soll die Software das leisten, was wir uns an Funktionalität vorgestellt haben. Erfüllt jede geschriebene Methode ihren Zweck, wird es einfacher, am bestehenden Projekt Veränderungen vorzunehmen. Nachstehend befindet ich eine Checkliste mit allen Funktionalitäten, die das Endprodukt aufweisen sollte.
Unsere Software arbeitet mit externen Systemen(z.B. slf4j) zusammen. Unserer Meinung nach sind diese auch geeignet. → Angemessenheit

Änderbarkeit
Hexagon (insbesondere das GUI) ist nach dem „Lego“-Prinzip aufgebaut. Diese Vorgehensweise macht es nicht nur einfacher, Veränderungen an unserem Spiel vorzunehmen, sondern mach den Code auch Verständlicher
Jede geschriebene Klasse wird ausführlich beschrieben und allfällige Errors werden (durch den Logger) dem Benutzer angezeigt

d) Messung der Qualitätsindikatoren

Es ist sehr schwierig, für die ausgewählten zwei Qualitätsmerkmale geeignete Skalen (Qualitätsmasse) zu erstellen, um diese zu Messen. Jedoch können wir Standards angeben (d.h. die Anforderungen, die wir an unser Spiel haben), welche durch einen konkreten Wert beschrieben werden können:

- Wir wollen, dass zum Schluss unseres Projektes mindestens 95% unserer geschriebenen Methoden in Javadoc erfasst sind!
- Wir wollen, dass unserere geschriebenen Methoden im Durchschnitt maximal dreissig Befehle (Kommentare und Leerzeilen sind damit nicht gemeint) umfassen. Dies steigert die Lesbarkeit unseres Codes!
- Wir wollen am Ende des Projekts mindestens 95% der untenstehenden Punkte mit Anforderungen an die Funktionalität erfüllen

Benutzbarkeit:

- Unser Spiel sollte ohne Anleitung spielbar sein
- Die Lobby sollte eher einfach gehalten werden, damit sich jeder Mitspieler darin zurechtfindet
- Die Steuerung im Spiel soll mit nur vier Richtungstasten erfolgen
  (Wie im MS1 vorgestellt)

Änderbarkeit:

- Von jeder Klasse sollen alle Methode so dokumentiert sein, dass jedermann Sinn und Zweck der Methode erkennen kann
- Soll z.B. ein Eckpunkt des Spielfeldes verschoben werden, soll dies mittels Änderung zweier Koordinaten fehlerfrei bewerkstelligt werden können

e) Messresultate Qualitätsmerkmal Änderbarkeit

Um das von uns gewählte Qualitätsmerkmal „Änderbarkeit“ zu messen (Qualitätsmasse), haben wir uns entschieden die Anzahl der Befehlszeilen pro Methode sowie die Anzahl der Methoden die mit Javadoc kommentiert sind, zu zählen/messen. Wir haben uns konkrete Messwerte zum Ziel gesetzt, die wir unterbieten wollen:

- jede Methode beinhaltet im Durchschnitt nicht mehr als 30 Befehlszeilen
- 95% der Methoden in unserem Code sind mit Javadoc kommentiert

Die erste Validierung unserer Qualitätsmasse am 28/29 April hat folgende Messwerte ergeben:

SVN-Revision: 1425	Messwert	Deutung des Wertes
Anzahl Methoden:	296	-
Anzahl Befehlszeilen:	2197	-
Durchsch. Anzahl Befehlszeilen:	7,422297297	Ein sehr erfreulicher Wert
Umfangreichste Methode	Methode „updateLobby“ mit 68 Befehlszeilen (Klasse: ClientMessageHandler)	Diese Methode ist zu lang
Anzahl Methoden kommentiert mit Javadoc	194 (65.54 %)	Hier besteht Aufholbedarf

Abb. 15: Ergebnisse von Messung Nr. 1

Für die zweite Validierung unserer Qualitätsmasse am 07/08/09 Mai wurden neu hinzugefügte Klassen und Methoden unseres Projektes nur teilweise in die Wertung mit einbezogen. Für die Anzahl der Methoden, der Befehlszeilen, sowie für die Durchschnittliche Anzahl Befehlszeilen wurde der JUnitTest miteinbezogen. Jedoch macht es keinen Sinn, die Methoden der Klasse LobbyTest mit Javadoc zu kommentieren, da deren Funktionsweide ziemlich selbsterklärend ist. Die Validierung hat folgende Messwerte ergeben:

SVN-Revision: 2151	Messwert	Deutung des Wertes
Anzahl Methoden:	294(inkl. Unit Test) 278(exkl. Unit Test)	-
Anzahl Befehlszeilen:	2099	-
Durchsch. Anzahl Befehlszeilen:	7,139455782	Ein sehr erfreulicher Wert
Umfangreichste Methode	Methode „updateLobby“ mit 68 Befehlszeilen (Klasse: ClientMessageHandler)	Diese Methode ist zu lang
Anzahl Methoden kommentiert mit Javadoc	184 (68.187 %)	Hier besteht Aufholbedarf

Abb. 16: Ergebnisse von Messung Nr. 2

f) Neues Tool für die Messungen

Um die Methoden und Befehlszeilen zu zählen, schauten wir bis jetzt in jede einzelne Klasse und zählten die benötigten Daten. Da diese Vorgehensweide sehr zeitintensiv ist, schauten wir nach einem Tool, das für uns diese Arbeit erledigt. Als dann endlich ein geeignetes Plug-In für Eclipse gefunden war, gab es Probleme bei der Installation. Deshalb konnten wir erst ab der dritten Messung davon profitieren. Hier ein Beispiel, wie sich eine Messung präsentiert (Gemessen am 9.5.16):

Number of Attributes (avg/max per type)	295
Number of Static Attributes (avg/max per type)	8
Number of Methods (avg/max per type)	338
src	322
Window	45
Server	43
hexagoal_game.game_logic.client	69
Client	28
Models	40
Server.User	21
hexagoal_game.game_logic.server	43
MessageHandling	25
Exceptions	8
Hexagon	0
tests	16
Number of Static Methods (avg/max per type)	3
Specialization Index (avg/max per type)	
Number of Classes (avg/max per packageFragment)	46
src	45
tests	1
Number of Interfaces (avg/max per packageFragment)	0
Number of Packages	11

Abb. 17: Der aufmerksame Leser hat sicherlich bemerkt, dass diese Messergebnisse (die wie jene von Messung zwei am 9.5.16 gemacht wurden), nicht mit den Resultaten der zweiten Messung übereinstimmen. Das Plug-In, „Metrics" genannt, zählt z.B auch die Anzahl der Konstruktoren zur Anzahl der Methoden (Gemäss der Beschreibung). Dies kann einfach nachgerechnet werden (am Beispiel der Methodenanzahl des Sourcefolders „src"):

Anz Methoden in „src". +1 weil Package „Hexagon" nicht dabei

Anz. Der Klassen (die alle mind. Einen Konstruktor enthalten)

$$323 - 45 = 278$$

Anz. Gleich wie bei der vorherigen Messung

Auch die Anzahl Befehlszeilen muss umgerechnet werden. Das Tool berechnet die gesamte Anzahl der Zeilen. Zählt also auch Zeilen die aus „{" oder „}" bestehen:

Total Lines of Code	4323
src	4125
tests	198

Abb. 18: Das Tool „Metrics" hat markant grössere Zahlen als Ergebnis bekommen, als wir

In der zweiten Messung wurde ein Wert von 2099 gemessen. Wie im einleitenden Text deklariert, wurden für die Messung beide Sourcefolder „src“ und „tests“ berücksichtigt. Für folgende Messungen werden wir die Messresultate des Plugins immer wie folgt verändern:

- $Anz.Methoden = (Messwert + 1) - 45$
- $Anz.Befehlszeilen = Messwert * \left(\frac{2099}{4323}\right) \approx Messwert * 0.485542$

SVN-Revision: 2663	Messwert	Deutung des Wertes
Anzahl Methoden:	571(inkl. Unit Test) 555(exkl. Unit Test)	Dieser starke Anstieg ist mit dem Hinzufügen von Get/Set-Methoden für jede Variable zu begründen
Anzahl Befehlszeilen:	2763	-
Durchsch. Anzahl Befehlszeilen:	4,838879159	Ein sehr erfreulicher Wert
Umfangreichste Methode	Methode „updateLobby“ mit 74 Befehlszeilen (Klasse: ClientMessageHandler)	Diese Methode ist zu lang
Anzahl Methoden kommentiert mit Javadoc	533 (96.03 %)	Eine erfreuliche Steigerung

Abb. 19: Die erste Messung nach der Änderung der Messmethode. Trotz der Umrechnung ist ein starker Anstieg in allen Bereichen festzustellen. (Zeitpunkt: 19.5.16)

g) Veränderungen während dem Projekt

Wie sich die Messwerte der definierten Qualitätsmasse des Qualitätsmerkmales „Änderbarkeit“ im Laufe der Zeit verändert haben, ist in folgenden drei Diagrammen festgehalten:

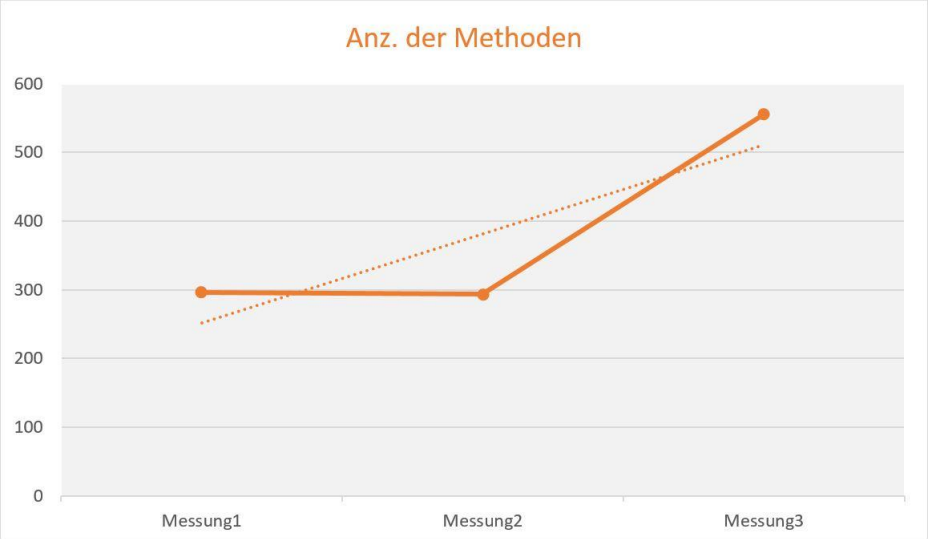


Abb. 20: Die Änderung den Anzahl geschriebener Methoden (inkl. Trendlinie)

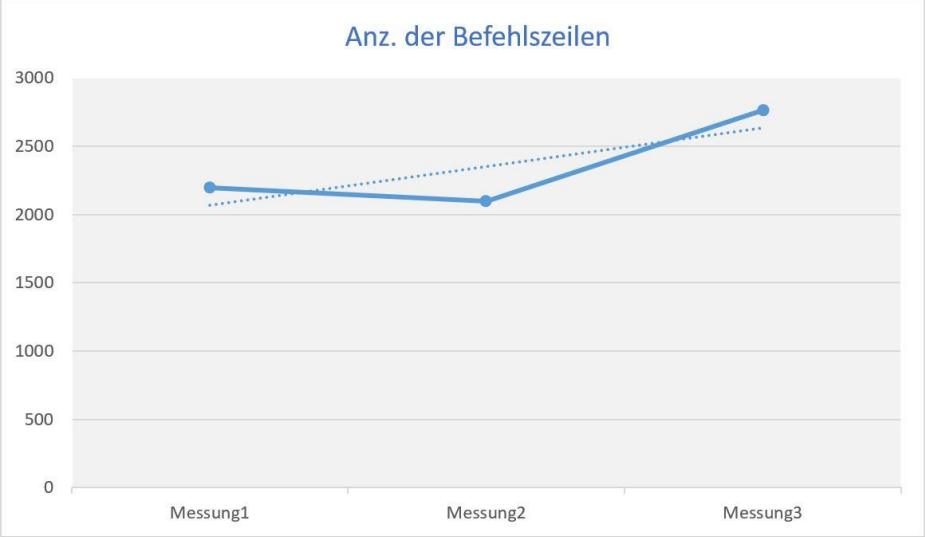


Abb. 21: Die Änderung den Anzahl geschriebener Befehlszeilen (inkl. Trendlinie)

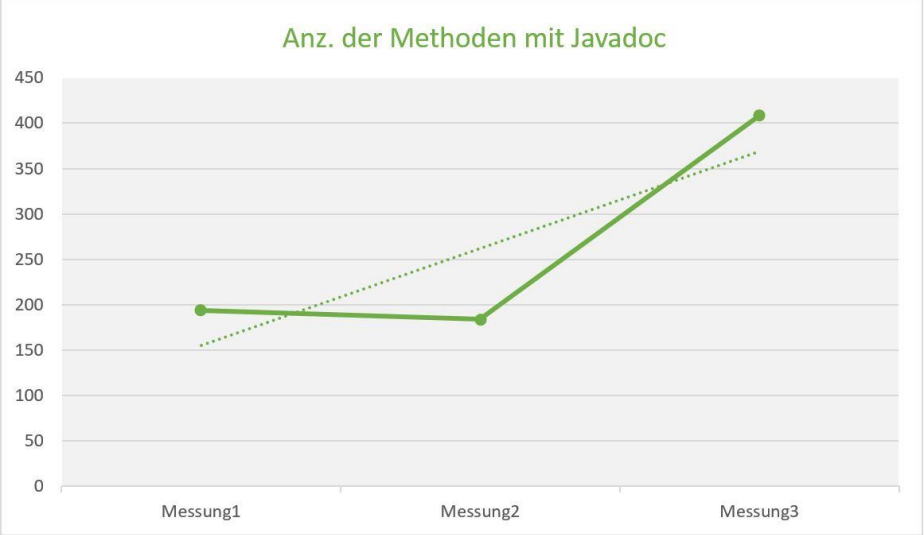


Abb. 22: Die Änderung den Anzahl mit Javadoc kommentierter Methoden (inkl. Trendlinie)

h) Messung Funktionalität

Unser zweites Qualitätsmerkmal „Funktionalität“ möchten wir anhand einer Checkliste messen. Vor und während der Programmierphase haben wir uns Gedanken darüber gemacht, was das Endprodukt alles können sollte. Zusammengefasst deckt die Liste folgende Themengebiete ab: Starten des Spiels, Plattformunabhängigkeit, Chat-Funktion, Übersichtlichkeit, Reconnect, Steuerung, Spielregeln und Verbindungsunterbruch. Auf der nächsten Seite finden Sie die vollständige Liste. Diese ist aber schon mit Kreuzen (✓) versehen, um zu markieren, dass dieser Punkt erfüllt ist:

- ○ Zum starten des Spiels genügt ein Mausklick
- ☑ Die Spielbarkeit/Ausführbarkeit des Spiels ist nicht vom verwendeten OS abhängig
  - Es ist auf Mac OS ausführbar
  - Es ist auf Windows ausführbar
  - Es ist auf Linux ausführbar
- ☑ In der Lobby kann man private Nachrichten (nur an ein Mitspieler) zu versenden
- ☑ In der Lobby kann man Nachrichten an alle Mitspieler zu versenden
- ☑ In der Lobby werden sowohl Spieler, als auch Games angezeigt
  - Es werden laufende und (noch) offene Spiele angezeigt
  - Man kann erkennen, welcher Spieler in welchem Game beigetreten ist
- ☑ In der Lobby ist es möglich, ein Game zu einem beliebigen Zeitpunkt zu starten
- ☑ Der Spieler kann einer beliebigen Lobby beitreten/diese verlassen
- ☑ Wenn ein Mitspieler die Verbindung verliert/ absichtlich unterbricht, wird dieser nach 10 Sekunden wieder ausgeblendet.
  - Findet allerdings ein Reconnect statt, wird der betreffende Spieler nicht aus der Lobby entfernt
- ☑ Es ist möglich, alte Spielstatiskiken einzusehen
- ☑ Ausserdem können Statistiken eines einzelnen Spielers eingesehen werden

Punkte zum eigentlichen Spiel:

- ☑ Jeder Spieler kann seinen „Punkt“ auf dem ganzen Spielfeld hin und her bewegen
- ☑ Jeder Spieler kann den Ball in jede beliebige Richtung stossen
- ☑ Der Ball prallt an den Banden
- ☑ Die Spieler werden als Punkte mit unterschiedlichen Farben dargestellt
- ☑ Die verbleibende Zeit wird korrekt angezeigt
- ☑ Für jeden Client ist der eigene Punkt sichtbar gekennzeichnet
- ☑ Nach jedem Torschuss wird neu angespielt (Anspiel in der Mitte)
- ☑ Ein Torschuss wird korrekt erkannt
  - Der Minuspunkt wird dem richtigen Spieler „abgezogen“
- ☑ Das Anspiel findet in der Mitte statt
- ☑ Die Spieler kollidieren untereinander
- ☑ Trennt ein Spieler die Verbindung während dem er in einem Game ist, wird das Spiel kurzzeitig eingefroren, und nach erfolgreichem Reconnect wieder fortgeführt

i) Deutung der Messresultate

Obwohl bei der Messung des Qualitätsmerkmales Änderbarkeit in allen Sparten eine deutliche Verbesserung der Messresultate zu verzeichnen war, haben wir unser Ziel, 95% aller geschriebenen Methoden mit Javadoc zu versehen nur (knapp) erreicht. Jedoch freuen wir uns, Ihnen mitteilen zu dürfen, dass wir uns damit nicht zufrieden gegeben haben, sondern fleissig weiter Javadoc Kommentare verfasst haben. Zum Zeitpunkt des 21 Mai 2016, drei Tage nach der letzten offiziellen Messung, kann man durch unsere Klassen „scrollen“, und findet keine unkommentierte Methode mehr.

Die Messung der Qualitätsmasse unseres zweiten Qualitätsmerkmals „Funktionalität“ ist hingegen sehr erfolgreich verlaufen. Fast alle Punkte, die wir zu Beginn des Projektes erfüllen wollten, konnten wir in die Realität umsetzen. Insgesamt konnten wir 20 von 21 Anforderungen gerecht werden. Umgerechnet in den prozentualen Anteil sind das: $\left(\frac{20}{21}\right) * 100 = 95.238$ mehr als die 95%, die wir uns zum Ziel gesetzt haben.

7. Dokumentation

Die gesamte Entwicklungsphase unseres Spiels haben wir in einem Blog festgehalten. Dieser Blog ist unter folgender URL abrufbar:

http://dreimannfussball.blogspot.ch