

Hausaufgabenblatt 2

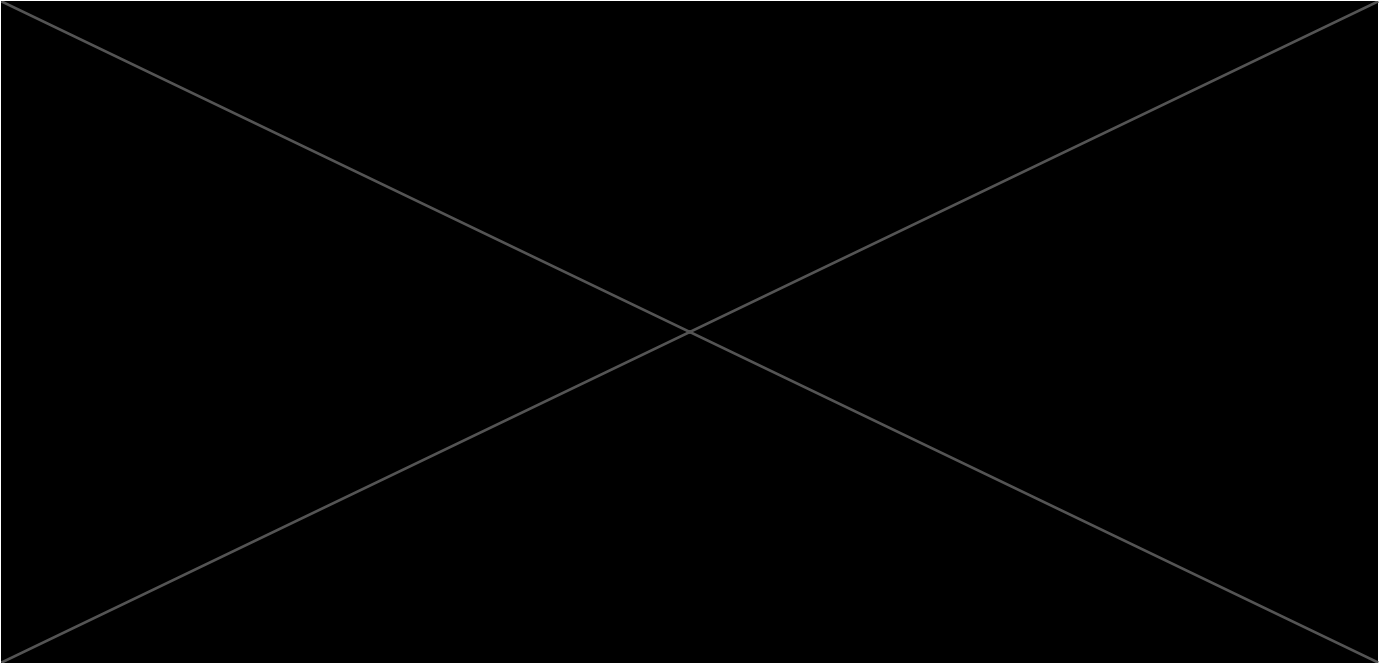


Im Rahmen dieser Hausaufgabe erweitert Ihr den generischen Spiele-Server aus der freiwilligen Hausaufgabe um das Spiel **Ploy** mit leichten Modifikationen. Alle notwendigen Informationen zur Implementierung des Spiels könnt Ihr diesem Blatt entnehmen. Eure Aufgabe besteht dabei aus drei Teilen:

Server: Der Gameserver verwaltet den Spielablauf und die Verbindung mit dem Web-Frontend. Seine Hauptaufgabe besteht darin, *Züge auf ihre Korrektheit zu überprüfen* und *den neuen Spielzustand zu berechnen* und auszuliefern. Eure Aufgabe ist die konkrete Implementierung der Spielregeln von **Ploy** innerhalb der Gameserver-Vorgabe. Die Implementierung erfolgt objekt-orientiert in **Java**.

Qualitätssicherung: Um einen korrekten Spielverlauf gewährleisten zu können, soll die von euch implementierte Funktionalität des Servers ausgiebig mit Hilfe von **JUnit** *getestet* werden. Außerdem soll die *Codequalität* mithilfe bestimmter Metriken gemessen und sichergestellt werden.

Bot: Da neben dem Spiel gegen einen menschlichen Herausforderer auch das Spiel gegen Computergegner möglich sein soll, entwickelt Ihr einen Bot in **Haskell**. Der Bot soll für gegebene, korrekte Spielzustände *alle möglichen Züge berechnen* und *einen aussuchen*. Es soll zum Schluss möglich sein, verschiedene Bots gegeneinander antreten zu lassen und so ein Turnier eurer Implementierungen zu veranstalten.



Hausaufgabenblatt 2



Im Rahmen dieser Hausaufgabe erweitert Ihr den generischen Spiele-Server aus der freiwilligen Hausaufgabe um das Spiel **Ploy** mit leichten Modifikationen. Alle notwendigen Informationen zur Implementierung des Spiels könnt Ihr diesem Blatt entnehmen. Eure Aufgabe besteht dabei aus drei Teilen:

Server: Der Gameserver verwaltet den Spielablauf und die Verbindung mit dem Web-Frontend. Seine Hauptaufgabe besteht darin, *Züge auf ihre Korrektheit zu überprüfen* und *den neuen Spielzustand zu berechnen* und auszuliefern. Eure Aufgabe ist die konkrete Implementierung der Spielregeln von **Ploy** innerhalb der Gameserver-Vorgabe. Die Implementierung erfolgt objekt-orientiert in **Java**.

Qualitätssicherung: Um einen korrekten Spielverlauf gewährleisten zu können, soll die von euch implementierte Funktionalität des Servers ausgiebig mit Hilfe von **JUnit** *getestet* werden. Außerdem soll die *Codequalität* mithilfe bestimmter Metriken gemessen und sichergestellt werden.

Bot: Da neben dem Spiel gegen einen menschlichen Herausforderer auch das Spiel gegen Computergegner möglich sein soll, entwickelt Ihr einen Bot in **Haskell**. Der Bot soll für gegebene, korrekte Spielzustände *alle möglichen Züge berechnen* und *einen aussuchen*. Es soll zum Schluss möglich sein, verschiedene Bots gegeneinander antreten zu lassen und so ein Turnier eurer Implementierungen zu veranstalten.

Hinweise zur Bewertung

Die Bewertung Eurer Abgabe erfolgt (soweit möglich) automatisch. Daher werden in erster Linie *funktionale Aspekte* Eurer Abgabe bewertet. Wir legen Wert auf eine faire Bewertung, das heißt, wir wollen sicherstellen, dass für alle Teilnehmer dieselben Bedingungen gelten und alle an dem selben Maßstab gemessen werden. Die faire Bewertung nicht-funktionaler Eigenschaften würde also eindeutig definierte Maßstäbe voraussetzen. Doch wie misst man die Schönheit von Code? Oder die Sinnhaftigkeit von unvollständigen Codefragmenten? Für einige Kriterien gibt es gute Metriken, deren Einhaltung wir unter dem Punkt Qualitätssicherung bewerten können. Für alles andere gilt, "sinnvoll" ist Code genau dann, wenn er der Erfüllung der Use-Cases zuträglich ist - was wir mit unseren Tests (und Ihr mit Euren) überprüfen.

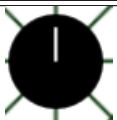



Viel Erfolg!

Ploy

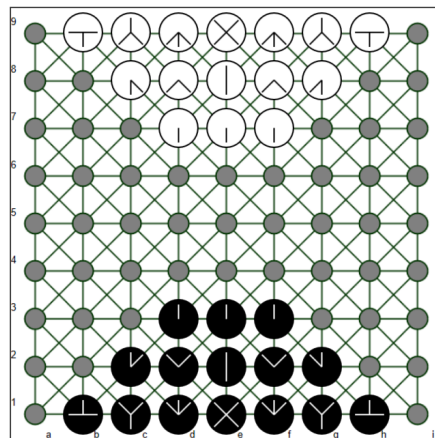
Ploy ist ein Brettspiel ähnlich Schach, in dem jeder Spieler eine Menge Figuren mit verschiedenem Bewegungsfreiraum besitzt, mit dem gegnerische Figuren geschlagen werden können. Ziel ist es, entweder den *Commander* oder alle anderen Figuren des Gegners zu schlagen. Es gibt dieses Spiel für zwei oder vier Spieler, wobei nur die Zwei-Spieler-Variante Teil der Aufgabe ist.

Regeln

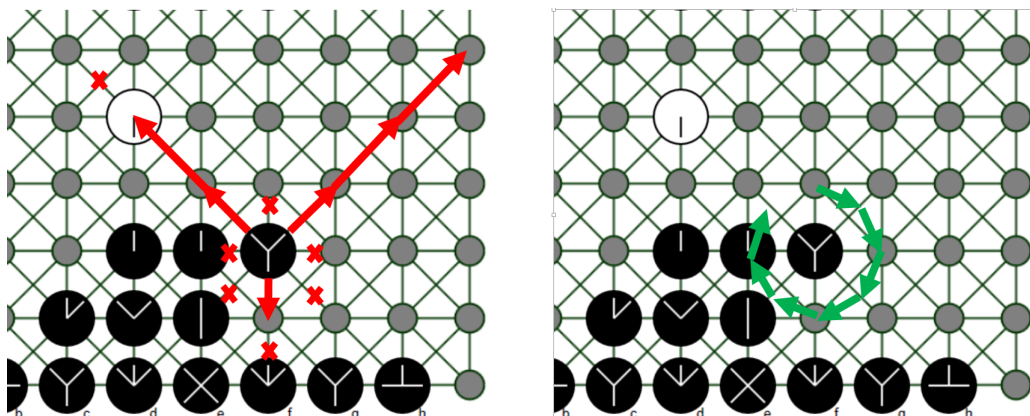
Gespielt wird auf einem zweidimensionalen Spielbrett mit 9x9 Feldern - schwarz beginnt. Jeder Spieler besitzt am Anfang 15 Figuren, auf denen unterschiedlich viele Bewegungsrichtungen durch Striche markiert sind. In jedem Zug kann eine Figur bewegt und/oder gedreht werden, wobei sich die Fähigkeiten der einzelnen Figuren unterscheiden. Konkret gibt es die folgenden Figuren:

Figur	Varianten	Distanz Bewegungen / Drehung
Shield		≤ 1 Feld und/oder Drehung (danach)
Probe		entweder bis zu 2 Felder oder Drehung
Lance		entweder bis zu 3 Felder oder Drehung
Commander		entweder ≤ 1 Feld oder Drehung

Das Startfeld sieht dabei wie folgt aus:



In jedem Zug muss eine Bewegung und/oder eine Drehung durchgeführt werden, mit dem eine Veränderung des Zustands herbeigeführt wird. Die Bewegung einer Figur ist horizontal, vertikal und diagonal entlang der auf der Figur abgebildeten Richtungslinien möglich. Dabei können Figuren nicht übersprungen werden. Im Zielfeld darf entweder keine Figur oder eine gegnerische Figur sein, wobei im zweiten Fall die gegnerische Figur vom Spielfeld entfernt wird. Die folgende Grafik zeigt die Zugmöglichkeiten der Y-Lance:



Das Ende des Spiels ist erreicht, wenn einer der beiden Spieler entweder keinen Commander mehr besitzt, oder nur noch den Commander. Derjenige Spieler, der den letzten Spielzug durchgeführt hat, gewinnt.

Notation

Für die Beschreibung der Spielsituation und Spielzüge verwenden wir eine abgewandelte FEN-Notation (bekannt von Schach). Das heißt, die Spalten werden mit kleinen Buchstaben von a bis i beschriftet, die Reihen mit Zahlen von 1 bis 9. Um ein Feld eindeutig zu bestimmen, wird immer zuerst die Spalte und dann die Zeile angegeben. Dabei bezeichnet a1 die untere linke Ecke, i9 die obere rechte. Zu Beginn stehen die schwarzen Figuren auf den “unteren” Feldern in den Reihen 1-3, die weißen auf den “oberen” in Reihe 7-9.

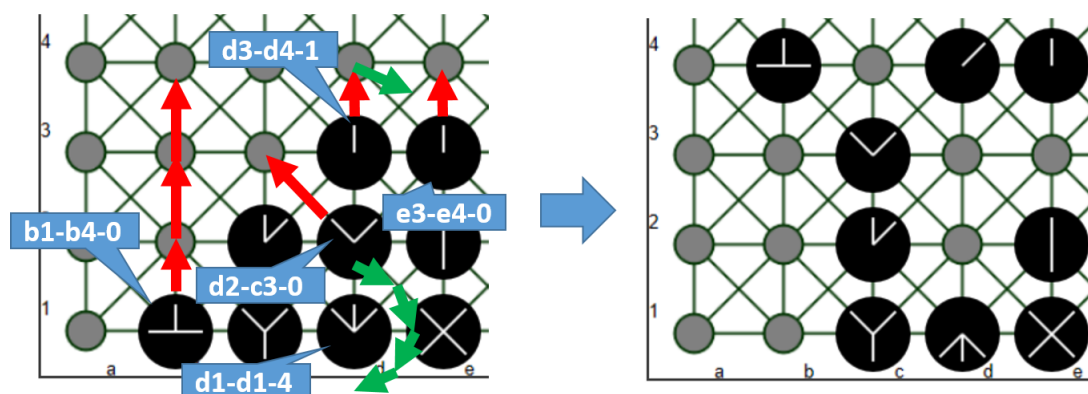
Das Spielbrett wird durch einen String beschrieben, der wie folgt aufgebaut ist: Reihen werden mit "/" getrennt, und innerhalb der Reihen werden die Felder mit "," getrennt. Am Anfang und am Ende des Strings gibt es kein "/", und am Anfang und Ende jeder Reihe auch kein ",". Zuerst wird das Feld a9 angegeben, zuletzt das Feld i1. Innerhalb eines Feldes kann eine Figur stehen, die wie folgt codiert ist: Das erste Zeichen ist die Farbe, "b" steht für schwarz und "w" steht für weiß. Die nachfolgenden Stellen sind eine Dezimalzahl zwischen 1 und 255. Diese codiert, als vorzeichenlose 8-bit Binärzahl aufgefasst, mit jedem Bit eine Richtung, in die Figur gehen darf (In der GUI als Strich dargestellt). Das LSB steht dabei für oben (bzw. Norden, oder entlang einer Spalte in Richtung der Reihe mit der höheren Zahl), und die jeweils höheren Bits stehen für die anderen Richtungen, im Uhrzeigersinn von Norden ausgehen. Beispiel: **b21**, also **0b00010101**, beschreibt die folgende Figur:



Das Spielbrett zu Beginn wird also wie folgt beschrieben:

,w84,w41,w56,w170,w56,w41,w84,/ ,w24,w40,w17,w40,w48,,/
 ,,w16,w16,w16,,,/, , , , , , , / , , , , , , , / , , , b1,b1,b1,,,/
 ,,b3,b130,b17,b130,b129,,,/ ,b69,b146,b131,b170,b131,b146,b69,

Züge werden durch einen String der Form `<start>-<ziel>-<drehung>` beschrieben, wobei `<drehung>` die Anzahl der Dreh-Schritte im Uhrzeigersinn beschreibt. Eine Drehung, die bewirkt, dass eine Linie statt nach Norden nun nach Süd-Osten zeigt, würde durch `<drehung>=3` beschrieben. Ein Schritt gegen den Uhrzeigersinn nach Nord-Westen entsprechend mit `<drehung>=7`. Drehungen um mehr als sieben Schritte sind nicht erlaubt. Entsprechend den bestehenden Möglichkeiten hat ein Zug also immer sieben Zeichen. Ein paar Beispiele:



1. Spielserver für Ploy in Java (12 Punkte)

Aufgabenbeschreibung

Der Großteil des Gameservers wurde bereits implementiert. Das abstrakte Modell aus Hausaufgabe 1 wurde, ähnlich wie in der Beispiellösung spezifiziert, bereits umgesetzt. Außerdem ist die GUI und die Kommunikation dorthin bereits gegeben. Die Aufgabe ist es nun, in der bereits angefangenen Klasse `PloyGame` für das konkrete Spiel Ploy ein Datenmodell zu entwickeln, mit dem der Spielzustand beschrieben werden kann. Unter Verwendung dieses Datenmodells soll die Funktion `tryMove(String moveString, Player player)` implementiert werden, die einen Zug prüft und ggf. durchführt. Konkret ist dabei folgendes zu tun:

- Ein Benutzer könnte das System angreifen oder schummeln, indem er die Anfragen an den Server manipuliert. Es ist zu prüfen, ob die Eingabe des Zuges im oben genannten Format erfolgt ist. Es ist außerdem zu prüfen, ob der übergebene Player an der Reihe ist.
- Es ist zu prüfen, ob der Zug nach den in diesem Aufgabenblatt gegebenen Regeln von Ploy gültig ist.
- Falls der Zug gültig ist, muss er durchgeführt werden, d.h. die Änderung muss in der internen Repräsentation des Spielfelds hinterlegt werden, sodass der geänderte Spielzustand zurückgegeben werden kann. Anschließend muss der andere Spieler am Zug sein. Falls das Spiel durch den Sieg des Spielers muss dies entsprechend in den Attributen der Klasse `Game` vermerkt werden. Außerdem muss die Spielhistorie gepflegt werden (s. Oberklasse `Game`).

Im Normalfall startet ein neues *PloyGame* mit dem Startzustand (siehe Ploy-Regeln).

Zusätzlich sind noch zwei Funktionen zu implementieren, mit denen ein Spielzustand (siehe Spielbrett-Repräsentation) übergeben (`setBoard(String)`) bzw. abgerufen `String getBoard()` werden kann. `setBoard()` dient zusammen mit der schon existierenden Funktion `setNextPlayer()` in unseren und euren Testfällen dazu, das Spiel an jedem beliebigen Punkt zu starten und so einzelne Situationen kompakter testen zu können. Als Testfunktion muss sie nicht die Eingaben auf Validität prüfen, alle unsere Eingaben können so im Spiel vorkommen.

Sämtlicher von euch eingefügter Code soll mit JUnit-Tests automatisiert getestet werden. Dabei soll 100% Zweigabdeckung (auf Bytecode-Ebene) erreicht werden. Testgegenstand ist dabei allerdings die Spezifikation - wenn also Funktionalität fehlt, sind die Testfälle entsprechend auch nicht vollständig. Nur diejenigen Testfälle werden bei der Messung der Testabdeckung gewertet, die auch erfolgreich durchlaufen.

Nach der Implementierung der Funktionalität und der Testfälle müssen eventuell noch geeignete Refactorings angewendet werden. Der von euch zu entwickelnde Code muss die folgenden Metriken erfüllen:

Metrik	Maximalwert
Zeilen pro Funktion (Method Lines of Code)	25
McCabe zyklomatische Komplexität	10
Verschachtelungstiefe (Nested Block Depth)	4
Anzahl Parameter pro Funktion	5

Vorgabe

Als Vorgabe haben wir euch ein Eclipse Servlet-Projekt zusammengestellt, in dem sämtliche Client-Funktionalität (HTML-Dateien mit Javascript) und das Servlet bereits enthalten sind. Im Projekt existiert auch eine README-Datei, die einige Informationen zur Orientierung enthält. In den schon vorhandenen Klassen markieren TODO-Kommentare die Stellen, an denen Ihr weitermachen sollt (Funktionalität fehlt in der Klasse `PloyGame`). Selbstverständlich fehlen auch noch Klassen. Neue Klassen müssen im Paket `de.tuberlin.sese.swtpp.gameserver.model.ploy` erstellt werden. Bestehender Code darf nicht verändert werden (außer wenn es explizit da steht).

Für die Testfälle haben wir ein spezielles Format definiert. Ein Beispieltestfall findet sich in der Klasse `TryMoveTest` (hier sollen auch eure Testfälle implementiert werden). Es stehen euch für die Testfälle drei Funktionen zur Verfügung:

- Die Funktion `startGame` sorgt dafür, dass das Spiel gestartet wird, und zwar zum einfacheren Testen mit einem beliebigen Spielstand (Figurenbelegung auf dem Board im FEN-String) und der Auswahl, wer an der Reihe sein soll.
- Die Funktion `assertMove` übergibt dem Spiel einen Spielzug und den ausführenden Spieler, und prüft, ob das Ergebnis eurem erwarteten Ergebnis entspricht (`tryMove` gibt `true` zurück, wenn der Spielzug gültig war und durchgeführt wurde).
- Die Funktion `assertGameState` prüft, ob sich der Gesamt-Spielzustand (aktuelles Board(FEN), nächster Spieler, Spiel beendet oder nicht, unentschieden, wer hat gewonnen) mit dem deckt, was ihr als Erwartung übergebt.

In euren Testfällen dürfen nur diese drei Funktionen mit explizit als String-Konstanten und Booleans übergebenen Argumenten (keine Variablen, siehe Beispieltestfall) verwendet werden. Jeder Testfall kann beliebig viele Aufrufe all dieser Funktionen enthalten - für eine hohe Abdeckung empfehlen sich aber sehr kurze Testfälle (Nur erfolgreiche Testfälle werden gewertet). Jeder Testfall muss mit einem Aufruf von `assertGameState` enden. Testfälle, die zusätzliche Funktionen aufrufen, werden NICHT berücksichtigt.

Hinweise

- In der Vorgabe ist eine simple Persistierung eingebaut, die erstellte Daten speichert und beim Neustart wieder lädt. Das soll euch das manuelle Testen über die GUI

erleichtern. Sollte es zu inkonsistenten Zuständen des Webserver kommen, so könnt ihr die Daten löschen, indem ihr die *test.db* Datei löscht. Diese wird mithilfe eines relativen Pfades in der Klasse *GameServerServlet* definiert. Neue Klassen sollten das Interface `java.io.Serializable` implementieren. Der Speicherort des Datenbank-Files kann in der Datei *GameServerServlet* angepasst werden.

- Wenn ihr den Bot (siehe Aufgabe 2) mit eurer Java-Implementierung ausprobieren wollt, könnt ihr den mit Hilfe der Klasse *HaskellBot* anbinden. Dazu müsst ihr den Bot zu einer ausführbaren Datei kompilieren (siehe *Main.hs*) und den Pfad, in dem diese zu finden ist, in der *GameFactory*-Klasse hinterlegen.
- Die Abdeckung wird von uns mit dem Plugin *Emma* in Eclipse geprüft. Wenn all euer Code in Emma grün angezeigt wird (100% Branch Coverage), ist ausreichende Testabdeckung erreicht. Die Abdeckung wird abzüglich der Anteile gewertet, die durch die Testfälle in der Vorgabe schon abgedeckt sind. Außerdem werden wir mit eigenen Testfällen vom gleichen Format prüfen, ob die Funktionalität vollständig implementiert ist. Bei fehlender Funktionalität fehlen also zusätzlich auch Testfälle.
- Die Vorgabe enthält einen Example-Test. Dieser ist vom selben Format wie unsere Testfälle. Wenn er also nicht durchläuft, stimmt also evtl. grundsätzlich etwas mit eurer Implementierung nicht, z.B. euer Parser des FEN-Strings.
- Die Metriken messen wir mit dem Eclipse-Plugin *Metrics 1.3.6* bzw. *1.3.8* (je nach Eclipse-Version). Dabei gilt auch hier: Nur bei ausreichend implementierter Funktionalität kann auch die volle Erfüllung der Metriken erreicht werden.
- Abzugeben ist das Projekt als exportiertes eclipse-Projekt im .zip-Format. Benennt auch das Projekt um, so dass es den Gruppennamen enthält. Weitere Infos zur Entwicklungs- und Ausführungsumgebung:
 - mindestens JDK 8 (JRE reicht für Tomcat nicht aus)
 - Eclipse (Vorgabe getestet mit 2018-09, lief aber auch schon mit früheren Versionen) in der J2EE-Version
 - Tomcat 9 <https://tomcat.apache.org/download-90.cgi>
 - Eine Anleitung und Tipps zur Einrichtung liegt dem Hausaufgabenblatt in der Datei *J2EE.Manual.pdf* auf ISIS bei.
 - Es kann sein, dass das Projekt nicht in jedem Browser läuft. Firefox sollte gehen, Javascript sollte dabei aktiviert sein. Der interne Browser von Eclipse geht nicht. Bitte beachtet, dass es für Tests mit mehreren Spielern nicht reicht verschiedene Tabs aufzumachen. Benutzt mehrere Browser, einen Spieler im Private Browsing oder eben verschiedene Computer (wofür das ja eigentlich gemacht ist).
 - Die direkte Verwendung von Code aus dem Internet ist nicht erlaubt und wird als Plagiat bewertet. Das Gleiche gilt für gruppenübergreifende Plagiate. Sollte

wir so etwas bei der Korrektur sehen, wird die Prüfung bei allen beteiligten Gruppenmitgliedern als "Nicht bestanden" gewertet. Das gilt natürlich für alle Teile der Abgabe.

- Um Spielfelder zu visualisieren könnt ihr die GUI Testseite verwenden. Ihr findet sie bei gestartetem Server unter http://localhost:8080/GameServer/Ploy_Board.html. Das anzuzeigende Spielbrett könnt ihr als JavaScript Parameter übergeben, zum Beispiel so:
http://localhost:8080/GameServer/Ploy_Board.html?w84,w41,w56,w170,w56,w41,w84,/,w24,w40,w17,w40,w48,/,w16,w16,w16,/,/,/,/,/,/,/,/,b69,,b2,b1,/,b130,,b1,/,b3,,b17,b130,b129,/,b146,b56,b170,b56,b146,b69

2. Ploy-Bot in Haskell (8 Punkte)

Ziel dieser Aufgabe ist es, einen Bot für Ploy in Haskell zu entwickeln. Eure Lösungen werden mit HUnit automatisch getestet. Das heißt, dass nur funktionale Eigenschaften überprüft werden – nicht-funktionale Eigenschaften wie Schönheit oder Komplexität des Codes werden dagegen nicht bewertet.

Auf ISIS findet Ihr eine Vorgabe zur Entwicklung eurer Lösung. Es gelten folgende Anmerkungen:

- Der Name des Moduls (`PloyBot`) **darf nicht verändert werden**.
- Alle Bibliotheken, die Haskell Platform mitliefert, können verwendet werden (Hinweis: Die Aufgabe lässt sich ohne weitere Imports lösen). Netzwerkzugriffe und die Einbettung von anderen Programmiersprachen sind nicht erlaubt. Deklarationen und Definitionen der `Util`-Funktionen **dürfen nicht verändert werden**.
- Die Signaturen und Namen der Funktionen `getMove` und `listMoves` **dürfen nicht verändert werden** (sie dienen der späteren Bewertung eurer Abgabe).

Eure Aufgabe besteht darin, die `getMove` und `listMoves` Funktionen sinnvoll zu vervollständigen. Beide erhalten als Argumente eine als String codierte Spielsituation, die sich aus dem Spielbrett (siehe Notation) und dem nächsten Spieler ("b" für schwarz und "w" für weiß) zusammensetzt. Ihr könnt davon ausgehen, dass ihr nur valide Strings erhaltet, die tatsächlich mögliche Spielsituationen codieren.

Die `listMoves`-Funktion soll eine Liste aller in der Situation möglichen Spielzüge berechnen und als String gemäß der Notation ausgeben. Die vollen Punkte erhaltet Ihr nur, wenn in allen getesteten Spielsituationen **alle regelkonformen Züge** korrekt berechnet werden. Dabei dürfen Züge **nicht mehrfach** vorkommen. Züge, die trotz unterschiedlicher Drehschritte die gleiche Änderung bewirken (bspw. Commander dreht um 1 bzw. 3 Schritte), zählen als unterschiedliche Züge und müssen vollständig aufgelistet werden.

Die `getMove`-Funktion wählt aus den möglichen Spielzügen einen aus und gibt nur diesen zurück. Das Auswählen geschieht nach eurer eigenen Strategie. Diese Funktion dient der Implementierung eines ausführbaren Bots für manuelle Tests mit der Java-Implementierung und für das Turnier. Die Implementierung dieser Funktion wird **nicht bewertet**.

In der Vorgabe findet Ihr außerdem eine Test-Datei, die eure Ausgaben auf korrekte Formatierung überprüft. Bitte reicht nur Abgaben mit korrekter Formatierung ein, da wir sonst keine Punkte vergeben können! Ihr startet die Tests, indem ihr das `Format`-Modul öffnet und `main` ausführt. Voraussetzung ist, dass Ihr HUnit¹ korrekt installiert habt.

Für die Abgabe ist nur die `PloyBot.hs` Datei interessant. Euer Bot wird unabhängig von eurem Server getestet.

¹<https://github.com/hspec/HUnit>