Technische Universität Berlin

Software and Embedded Systems Engineering Group
Prof. Dr. Sabine Glesner
www.sese.tu-berlin.de Sekr. TEL 12-4 Ernst-Reuter-Platz 7 10587 Berlin

Softwaretechnik und Programmierparadigmen WiSe 2016/2017

Prof. Dr. Sabine Glesner
Joachim Fellmuth joachim.fellmuth@tu-berlin.de
Tobias Pfeffer tobias.pfeffer@tu-berlin.de
Tutoren

Hausaufgabenblatt 2 **Ausgabe: 18.1.** **Abgabe: 15.2.**

Im Rahmen dieser Hausaufgabe erweitert Ihr den generischen Spiele-Server aus der freiwilligen Hausaufgabe um das Spiel Lasca (auch Laska, Lasker oder Laskers). Alle notwendigen Informationen zu dem Spiel könnt Ihr diesem Blatt entnehmen. Eure Aufgabe besteht dabei aus drei Teilen:

1. Der Spieleserver 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 die konkrete Implementierung des Spiels Lasca innerhalb des Spieleservers in Java.

2. Qualitätssicherung: Um einen korrekten Spielverlauf gewährleisten zu können, soll die von euch implementierte Funktionalität des Servers ausgiebig mit Hilfe von jUnit getestet werden. Außerdem sollen Metriken gemessen und ggf. durch Refactorings nachgebessert werden.

3. Ein Bot in Haskell. Da auch das Spiel gegen Computergegner möglich sein soll, entwickelt Ihr einen Bot, der für korrekte Spielzustände alle möglichen Züge berechnet und einen aussucht.

Es soll zum Schluss möglich sein, verschiedene Bots gegeneinander antreten zu lassen und so ein Turnier eurer Implementierungen zu veranstalten. Wie viel Intelligenz Ihr euren Bots spendiert ist allerdings euch überlassen, bewertet wird nur, dass alle ausgegebenen Züge zu dem jeweiligen Spielzustand passen.

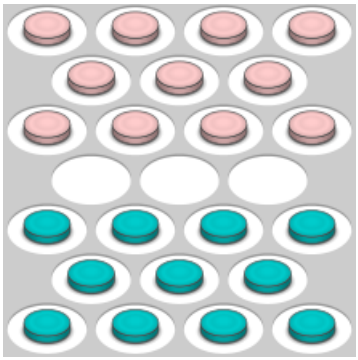
Viel Erfolg!

Lasca

Lasca wurde 1911 vom deutschen Schachweltmeister Emanuel Lasker entwickelt und ist ein Brettspiel für zwei Spieler. Es ähnelt dem Spiel Dame, gefangene gegnerische Spielsteine werden allerdings nicht vom Brett entfernt, sondern unter dem Fänger gestapelt. Andersrum können gefangene Steine wieder befreit werden und gelegentlich werden die "Soldaten" zu mächtigeren "Offizieren" befördert...

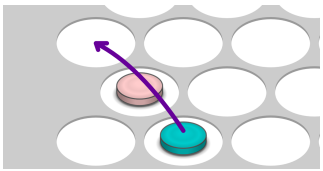
Regeln

Gespielt wird auf einem auf 7 x 7 Felder verkleinerten Schachfeld mit je 11 Spielsteinen in zwei Farben. Dabei wird immer nur jedes zweite Feld verwendet, also nur die weißen (oder schwarzen) Felder.

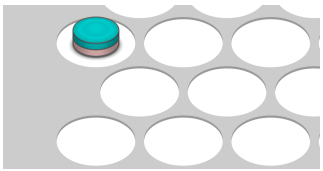


Startposition[1]

Normale Spielsteine (Soldaten) können nur diagonal nach vorne laufen und schlagen. Ein Spielstein kann auf ein angrenzendes Feld laufen, wenn dieses frei ist. Er kann schlagen, wenn ein angrenzendes Feld von einem gegnerischen Spielstein belegt wird und dieser übersprungen werden kann (das Feld diagonal dahinter also frei ist). Beim Schlagen wird der oberste Spielstein vom übersprungen Feld genommen und unten an den eigenen Spielstein angefügt. Wenn es möglich ist einen gegnerischen Spielstein zu schlagen, *muss* geschlagen werden.



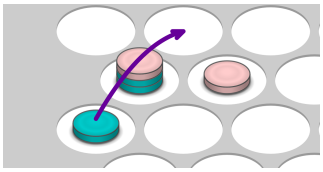
(a) Blau schlägt Pink[2]



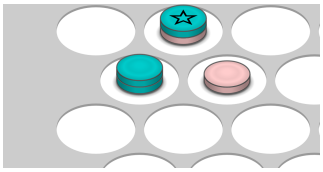
(b) Pink gefangen[3]

Erreicht ein Spielstein (durch schlagen oder laufen) den gegenüberliegenden Rand, wird er zum Offizier befördert. Offiziere können, anders als Soldaten, auch nach hinten laufen und schlagen. Ein begonnener Angriff *muss* fortgesetzt werden, falls durch denselben Spielstein

weitere Spielsteine geschlagen werden können. Dabei darf nicht direkt in die Gegenrichtung geschlagen werden (siehe Notation). Wird ein Spielstein während eines Angriffes zum Offizier befördert, kann der Angriff *nicht* weiter fortgeführt werden.



(a) Erreicht den Rand[4]

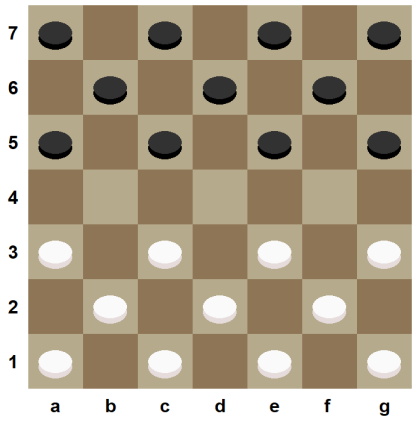


(b) Und wird promoviert[5]

Ein Spieler gewinnt das Spiel, wenn (a) der Gegner keine Spielsteine mehr kontrolliert, (b) der Gegner keinen gültigen Zug machen kann oder (c) der Gegner aufgegeben hat.

Notation

Für die Notation verwenden wir (mehr oder weniger) die Regeln von `http://www.pjb.com.au/laska/index.html#Notation`. Das heißt, die Reihen werden mit Zahlen von 1 bis 7 beschriftet, die Spalten mit kleinen Buchstaben von a bis g, wobei a1 die untere linke Ecke bezeichnet, g7 die obere rechte. Zu Beginn stehen die weißen Figuren auf den "unteren" Feldern von a1 bis g3, die schwarzen auf den "oberen" von a5 bis g7.



Bezeichnung der Felder

Bretter werden durch einen FEN-String beschrieben, wobei folgendes gilt:

b	Schwarzer Soldat	B	Schwarzer Offizier
w	Weißer Soldat	W	Weißer Offizier
,	Nächste Spalte	/	Nächste Reihe

[5]Bilder von Wikimedia/Cmglee - Own work, CC BY-SA 4.0

Zuerst wird das Feld a7 angegeben, zuletzt das Feld g1. Spielsteine werden von oben nach unten angegeben. Das Brett zu Beginn wird also durch
`b,b,b,b/b,b,b/b,b,b,b/,,/w,w,w,w/w,w,w/w,w,w,w` beschrieben.

Um den Spielzustand vollständig zu beschreiben, folgt auf die Beschreibung des Brettes die Angabe welcher Spieler zur Zeit an der Reihe ist. Zu Beginn (weiß fängt an) also:
`b,b,b,b/b,b,b/b,b,b,b/,,/w,w,w,w/w,w,w/w,w,w,w w`

Züge werden durch Start- und Zielposition beschrieben (z.B. von a1 nach b2: `a1-b2`). Wird ein Spielstein geschlagen, werden ebenfalls *nur Start- und Zielposition* beschrieben (z.B. von c5 über d4 nach e3: `c5-e3`. **Nicht:** `c5-d4-e3`). Weitere Schritte werden bei uns *nicht* direkt mit angegeben, sondern in einzelne Schritte zerlegt. Dafür wird in diesem Fall der letzte Zug mit in die Situationsbeschreibung aufgenommen. Am Ende des Aufgabenblattes findet Ihr ein kurzes Beispiel.

Ihr könnt euch auf dieser Seite mit dem Spiel und der Notation vertraut machen: `http://www.pjb.com.au/laska/play_laska.html`

Weitere Informationen

`http://www.lasca.org/` Regel und Infos (Englisch)

`http://www.pjb.com.au/laska/` Regeln und Infos (Englisch)

`http://www.pjb.com.au/laska/play_laska.html` Onlinespiel gegen den Computer

`http://ascal.sourceforge.net/` Implementierung in C++ für Linux

1. Spielserver für Lasca in Java (12 Punkte)

Aufgabenbeschreibung

Der Großteil des Spieleservers 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 LascaGame für das konkrete Spiel Lasca ein Datellmodell zu entwickeln, mit dem der Spielzustand beschrieben werden kann. Damit soll die Funktion tryMove (Format eines Zuges siehe Lasca-Notation) implementiert werden, die prüft ob ein Spielzug gültig ist, und ihn ausführt wenn möglich. Außerdem sind noch zwei Funktionen zu implementieren, mit denen ein Spielzustand (FEN-String siehe Abschnitt Lasca-Notation) übergeben bzw. angerufen werden kann. Im Normalfall muss ein neues *LascaGame* mit dem Startzustand (siehe Lasca-Regeln) starten.

Sämtlicher von euch eingefügter Code soll mit jUnit-Tests automatisiert getestet werden. Dabei soll Zweigabdeckung (auf Bytecode-Ebene) erreicht werden. Testgegenstand ist dabei allerdings die Spezifikation, wenn Funktionaltät fehlt, sind die Testfälle entsprechend auch nicht vollständig. Die Testfälle müssen 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 LascaGame). Selbstverständlich fehlen auch noch Klassen. Neue Klassen müssen im Paket `de.tuberlin.sese.swtpp.gameserver.model.lasca` erstellt werden. Bestehender Code darf nicht verändert werden (Außer wenn es exlizit da steht).

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

- Die Funktion `startGame` sorgt dafür, das das Spiel gestartet wird, und zwar zum einfacheren Testen mit einem Beliebigen Zustand/FEN String (Figurenbelegung auf dem Board) und der Auswahl, wer der Startspieler ist.

- 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, wer hat gewonnen) mit dem deckt, was ihr als Erwartung übergebt.

In euren Testfällen dürfen nur diese Funktionen vewendet werden. Jeder Testfall kann beliebig viele Aufrufe aller Funktionen enthalten. `assertMove` kann z.B. mehrfach verwendet werden, um einen Spielstand mit Move-History zu erzeugen. Jeder Testfall muss mit einem Aufruf von `assertGameState` enden. Testfälle müssen erfolgreich getestet werden sonst werden sie nicht in die Messung der Testabdeckung mit einbezogen.

Hinweise

- In der Vorgabe ist eine simple Persistierung eingebaut, die erstellte Daten speichert und beim Neustart wieder lädt. Das soll euch das manuelle Testen erleichtern. 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 3) 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 Bot-Klasse hinterlegen.

- Die Abdeckung 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. Außerdem werden wir mit eigenen Testfällen vom gleichen Format prüfen, ob die Funktion vollständig implementiert ist. Bei fehlender Funktionalität fehlen also auch Testfälle.

- Die Metriken messen wir mit dem Eclipse-Plugin Metrics 1.3.6. 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. Bitte benennt die Pakete um: von `swtpp` nach `swtpp.<gruppenname>`. Benennt auch das Projekt um so dass es den Gruppennamen enthält.
  Entwicklungs- und Ausführungsumgebung:
  - JDK 8 (JRE reicht für Tomcat nicht aus)

- Eclipse (Vorgabe getestet mit Mars) in der J2EE-Version
- Tomcat 8 `http://tomcat.apache.org/download-80.cgi`
- Eine Anleitung und Tipps zur Einrichtung haben wir euch in der Datei J2EE_Manual.pdf vorbereitet, die diesem Hausaufgabenblatt auf ISIS beiliegt.
- Es kann sein dass das Projekt nicht in jedem Browser läuft. Firefox sollte gehen, Javascript sollte dabei aktiviert sein.
- 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.

2. Lasca Bot in Haskell (8 Punkte)

Ziel dieser Aufgabe ist es, einen Bot für das Lasca-Spiel 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 (`LascaBot`) **darf nicht verändert werden**.
- Außer dem bereits importierten Modulen `Util` und `Data.Char` dürfen **keine weiteren Module** importiert werden.
- Die Signaturen und Namen der Funktionen `getMove` und `listMoves` **dürfen nicht verändert werden** (sie dienen der späteren Bewertung eurer Abgabe).
- Deklarationen und Definitionen der `Util`-Funktionen **dürfen nicht verändert werden**.

Eure Aufgabe besteht darin, die `getMove` und `listMoves` Funktionen zu vervollständigen. Beide erhalten als Argument eine als String codierte Spielsituation (siehe Notation). Ihr könnt davon ausgehen, dass ihr nur valide Strings erhaltet.

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.

Die `getMove`-Funktion wählt darüber hinaus 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) und wird **nicht bewertet**.

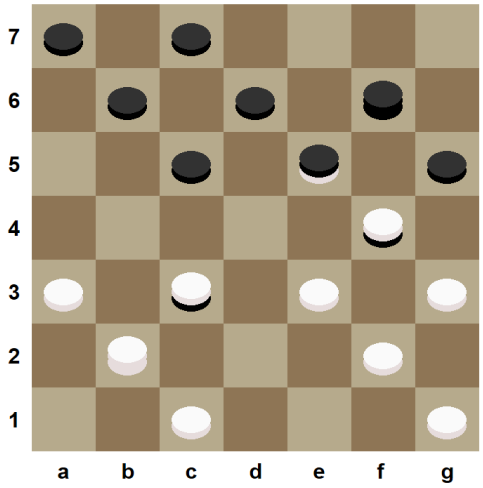
In der Vorgabe findet Ihr außerdem eine Test-Datei, die eure Ausgaben auf korrekte Formatierung überprüft. Bitte reicht nur Abgaben mit korrekter Formatierung ein, da wir sonst keine Punkte vergeben können! Um ein Gefühl für die Richtigkeit eurer Lösung zu erhalten sind außerdem ein paar Beispieltestfälle mit korrekten Zügen enthalten. Dieses Format werden wir auch verwenden. Ihr startet die Tests, indem ihr das `LascaTest`-Modul öffnet und entweder `runTestTT format` oder `runTestTT game` ausführt. Voraussetzung ist, dass Ihr HUnit[6] korrekt installiert habt.

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

[6] `https://github.com/hspec/HUnit`

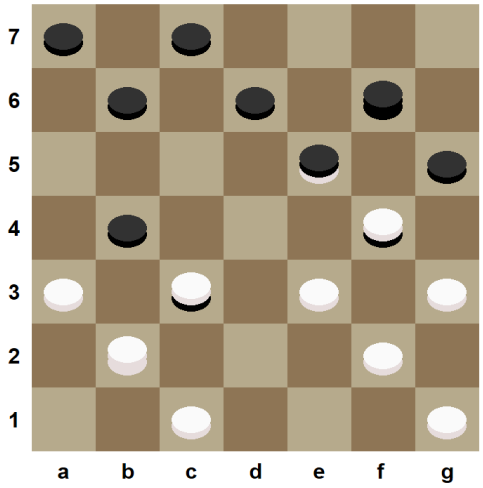
Beispiel

Ausgangssituation: `b,b,,/b,b,bb/,b,bw,b/,,wb/w,wb,w,w/ww,,w/,w,,w b`



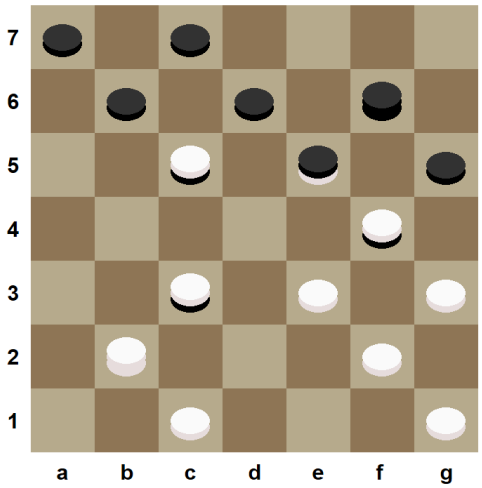
Mögliche Züge: `b6-a5, c5-b4, c5-d4, e5-d4`

Nach `c5-b4`: `b,b,,/b,b,bb/,,bw,b/b,,wb/w,wb,w,w/ww,,w/,w,,w w`



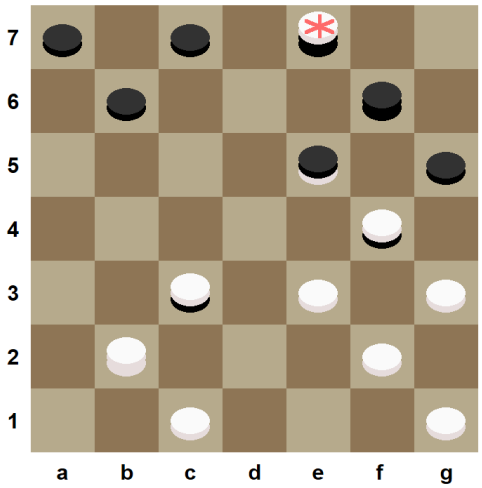
Mögliche Züge: `a3-c5, c3-a5`

Nach a3-c5: b,b,,/b,b,bb/,wb,bw,b/,,wb/,wb,w,w/ww,,w/,w,,w w a3-c5



Mögliche Züge: c5-e7 (Fortsetzung des Angriffs im zweiten Schritt)

Nach c5-e7: b,b,Wbb,/b,,bb/,,bw,b/,,wb/,wb,w,w/ww,,w/,w,,w b



Weiß wurde promoviert, schwarz ist am Zug