Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2024/25

Abschlussaufgabe 1
Version 1.4

20 Punkte

Ausgabe:	12.02.2025, ca. 12:00 Uhr
Abgabestart:	26.02.2024, 12:00 Uhr
Abgabefrist:	13.03.2024, 06:00 Uhr

Neuerungen

Version 1.1 Beispielinteraktion korrigiert (A.7.2, Z. 187ff.)

Version 1.2 Beschreibung der Grammatik korrigiert (A.5)

Version 1.3 Beschreibung des Zustands angepasst (A.2.3), Beschreibung von Schutz erweitert (A.2.4), Format der Konfiguration angepasst (A.5), Beispielinteraktion korrigiert (A.7.2, Z. 135)

Version 1.4 A.4.4.2 Beispielinteraktion korrigiert (Z. 4ff.), Ausgabe korrigiert (A.6 (2)), Vergabe der Wettstreiter-Nummer korrigiert (A.4.3.3), Ende des Schutzes hinzugefügt (A.2.7)

Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki

Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Abschlussaufgabe 1 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 17*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.nio.file`, `java.nio.charset`, `java.util.random`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 140 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln ein.

[2] `https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ`
[3] `https://sdq.kastel.kit.edu/programmieren/`
[4] `https://artemis.praktomat.cs.kit.edu/`

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Orientieren Sie sich zudem an den Bewertungskriterien im Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 26.02.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 13.03.2024, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

Prüfungsmodus in Artemis

Wenn Sie mit einer Abschlussaufgabe fertig sind, können Sie diese frühzeitig abgeben. Dazu dient Schaltfläche „Vorzeitig abgeben“. Nach der frühzeitigen Abgabe einer Abschlussaufgabe können Sie keine Änderungen an Ihrer Abgabe mehr vornehmen.

Aufgabe A: Monster-Wettstreit

A.1 Einführung

In dieser Aufgabe implementieren Sie ein rundenbasiertes Spiel, in dem Monster in Kämpfen gegeneinander antreten und versuchen, sich gegenseitig zu besiegen. Es beinhaltet u.a. ein einfaches Typsystem für Monster und ihre Aktionen, Zustandsbedingungen, Stärkung und Schwächung und ermöglicht somit komplexe Interaktionen. Die Monster und ihre Aktionen sind dabei nicht vorgegeben, sondern werden aus einer Eingabedatei eingelesen.

Im Folgenden werden zunächst die vorkommenden Konzepte kurz erklärt. Im Anschluss werden die geforderten Befehle, die das von Ihnen implementierte System unterstützen soll, mit beispielhaften Ein- und Ausgaben beschrieben.

A.2 Konzepte

Im Folgenden werden alle für diese Aufgabe relevanten Konzepte erläutert.

A.2.1 Monster

Monster sind die Akteure eines jeden Wettstreits. Jedes Monster verfügt über einen Namen sowie über ein *Element* und einen *Grundzustand*, die zusammen die individuellen Stärken und Schwächen eines Monsters definieren. Zudem beherrscht ein Monster eine bis vier *Aktionen*.

A.2.2 Grundzustand

Der Grundzustand eines Monsters fasst folgende Grundzustandswerte zusammen:

Maximaler Gesundheitswert (HP) Gibt die „volle Gesundheit“ an.

Angriffswert (ATK) Beeinflusst den Schaden, den eigene Aktionen bei Gegnern verursachen.

Verteidigungswert (DEF) Beeinflusst den Schaden, den gegnerische Aktionen verursachen.

Geschwindigkeitswert (SPD) Beeinflusst die Reihenfolge, in der die Monster am Zug sind.

Zielsicherheitswert (PRC) Beeinflusst die Trefferrate eigener Aktionen auf ein Monster.

Wendigkeitswert (AGL) Beeinflusst die Trefferrate gegnerischer Aktionen auf ein Monster.

Der Grundzustand eines Monsters ist unveränderlich und stellt den initialen *Zustand* für ein Monster in einem Wettstreit dar. Während eines Wettstreits können allerdings Aktionen von Monstern *Zustandsveränderungen* bewirken. Alle Grundzustandswerte sind positiv und ganzzahlig. Die Grundzustandswerte für Zielsicherheit und Wendigkeit der Monster sind nicht individuell, sondern haben grundsätzlich den Wert 1.

A.2.3 Zustand

Der Zustand eines Monsters in einem Wettstreit fasst folgende Zustandswerte zusammen:

Gesundheitswert Gibt die aktuelle Gesundheit an. Die Gesundheit kann durch Schaden sinken (aber nie unter 0) oder durch Heilung steigen (aber nie über den maximalen Gesundheitswert). Sinkt die Gesundheit auf 0, gilt das Monster als besiegt, ist kampfunfähig und scheidet aus dem Wettstreit aus.

Zustandsveränderungen Für die Grundzustandswerte Angriffswert, Verteidigungswert, Geschwindigkeitswert, Zielsicherheitswert und Wendigkeitswert werden während eines Wettstreits Änderungen nicht absolut, sondern in ganzzahligen *Stufen* von −5 bis +5 angegeben. Der Grundzustand entspricht dabei einer Zustandsveränderung von 0 Stufen. Die Zustandsveränderung von n Stufen wirkt sich durch folgenden multiplikativen Faktor auf den Grundzustandswert aus:

$$\text{statFactor(b,n)} = \begin{cases} \dfrac{b+n}{b}, & \text{wenn } n \geq 0 \\ \dfrac{b}{b-n}, & \text{wenn } n < 0 \end{cases}, \text{ wobei } b = \begin{cases} 2, & \text{wenn } stat \in \{ATK, DEF, SPD\} \\ 3, & \text{wenn } stat \in \{PRC, AGL\} \end{cases}$$

Zustandsbedingung Im Wettstreit kann ein Monster zu jedem Zeitpunkt unter höchstens einer *Zustandsbedingung* leiden. Solange ein Monster an einer Zustandsbedingung leidet, treffen *Zustandsbedingung*-Auswirkungen von Aktionen dieses Monster nicht. Insbesondere kann die Zustandsbedingung also nicht von der Auswirkung einer Aktion durch eine andere Zustandsbedingung überschrieben werden. Die Arten von Zustandsbedingungen und ihre Auswirkungen werden in Abschnitt A.2.5 erklärt.

Im Folgenden wird der Zustandswert, der Grundzustandswert sowie ggf. Zustandsveränderung und Zustandsbedingung miteinbezieht, als der *effektive* Zustandswert bezeichnet. Die effektiven Zustandswerte sollen ~~den Wert 1.0 nicht unterschreiten, und ansonsten~~ nicht gerundet werden.

Beispiel: Das Monster `GlitterUnicorn`, welches einen Grundverteidigungswert von 59 und eine aktuelle Zustandsveränderung von −2 für die Verteidigung hat und *klitschnass* ist, hat einen effektiven Verteidigungswert von $59 * \frac{2}{2+2} * \frac{3}{4} = 59 * \frac{3}{8} = 22.125$.

A.2.4 Aktion

Ist ein Monster in einem Wettstreit am Zug, kann es eine Aktion ausführen oder *passen*. Eine Aktion hat ein *Element* und eine geordnete Liste von *Auswirkungen*. Ein Monster kann eine bis vier Aktionen beherrschen, jedoch ist es unzulässig, dass ein Monster dieselbe Aktion mehrfach beherrscht. Die Arten von Auswirkungen sind vorgegeben, können aber beliebig kombiniert und durch Argumente angepasst werden.

Schaden (Parameter: Ziel[Anwendendes oder getroffenes Monster], *Stärke*, *Trefferrate*.) Falls diese Auswirkung trifft, nimmt das *Ziel*-Monster Schaden, d.h. der Gesundheitswert des getroffenen Monsters sinkt[10]. Die Größe des entstandenen Schadens hängt von der *Stärke* ab. Schaden durch gegnerische Auswirkungen kann durch Schutz verhindert werden[17c].

Zustandsbedingung (Parameter: Ziel[Anwendendes oder getroffenes Monster], *Zustandsbedingung*, *Trefferrate.*) Falls die Auswirkung trifft und das *Ziel*-Monster nicht bereits unter einer Zustandsbedingung leidet, erleidet das *Ziel*-Monster die angegebene *Zustandsbedingung*[12a 13a 14a 15a].

Zustandsveränderung (Parameter: Ziel[Anwendendes oder getroffenes Monster], *Zustandswert*, *Differenz*, *Trefferrate.*) Falls die Auswirkung trifft, wird das *Ziel*-Monster von einer Zustandsveränderung getroffen, die den angegebenen *Zustandswert* um eine *Differenz* verringert oder erhöht[16a 16b], z.B. +2 Stufen, −1 Stufe. Verringerungen von Zustandsveränderungen durch gegnerische Auswirkungen können durch Schutz verhindert werden [17d].

Schutz (Parameter: *Ziel*[Gesundheit oder Zustandswerte], *Anzahl Runden*, *Trefferrate.*) Falls die Auswirkung trifft, ist das ausführende Monster ab sofort bis zum Ende der Runde und dann für die ganze Dauer der gegebenen *Anzahl Runden* gegen bestimmte Veränderungen an seinem Zustand immun[17a 17b]. Diese Veränderungen werden durch das *Ziel* der Auswirkung bestimmt.

Schutz der Gesundheit Die Gesundheit des ausführenden Monsters nicht durch direkte Auswirkungen gegnerischer Aktionen gesenkt werden; Schaden an sich selbst, Heilung sowie Brandschaden sind jedoch weiterhin möglich.

Schutz der Zustandswerte Die Zustandswerte *ATK*, *DEF*, *SPD*, *PRC*, *AGL* können alle nicht durch Auswirkungen gegnerischer Aktionen gesenkt werden; Senken durch eigene Aktionen sowie Erhöhung sind weiterhin möglich. Bestehen zum Zeitpunkt, zu dem der Schutz einsetzt, bereits Zustandsänderungen, werden diese durch den Schutz nicht rückgängig gemacht und sind weiterhin wirksam. Des Weiteren sind Zustandsänderungen durch Zustandsbedingungen weiterhin wirksam.

Falls beim anwendenden Monster bereits ein Schutz besteht, wird dieser ohne Meldung beendet und durch den neuen Schutz ersetzt. Dabei ist es unbedeutend, ob der bestehende und der neue Schutz das gleiche *Ziel* haben oder nicht.

Heilung (Parameter: *Ziel*[Anwendendes oder getroffenes Monster], *Stärke*, *Trefferrate.*) Falls die Auswirkung trifft, erhält das *Ziel*-Monster Gesundheit zurück[11], invers zu *Schaden*.

Wiederholung (Parameter: *Anzahl*, mindestens eine *Auswirkung.*) Die angegebenen *Auswirkungen* werden entsprechend der *Anzahl* oft wiederholt. Anmerkung: Wiederholungen innerhalb von Wiederholungen sind nicht zulässig.

Fortfahren (Parameter: *Trefferrate.*) Falls diese Auswirkung trifft, geschieht nichts und es wird mit der nächsten Auswirkung fortgefahren. Anmerkung: Mit dieser Auswirkung können Sie die Trefferrate einer Aktion steuern, wenn diese sich von der Trefferrate der ersten „eigentlichen“ Auswirkung unterscheiden soll.

Die Trefferraten sind dabei jeweils in Ganzzahlen zwischen 0 und 100 angegeben und stehen für Prozentzahlen. Die Trefferrate einer Auswirkung einer Aktion betrifft jeweils nur diese Auswirkung. Eine Ausnahme hierzu stellt die erste Auswirkung einer Aktion dar: trifft diese nicht, gilt die gesamte Aktion als fehlgeschlagen.

Für die unterstrichenen Parameter kann eine gleichverteilte, ganzzahlige Zufallszahl in einem festen Intervall verwendet werden, sodass sie in jeder Ausführung (und ggf. jeder Wiederholung) u.U. verschieden sind.

Für die *Stärke* von Schaden und Heilung gibt es drei Arten von Angaben:

Basisstärke (Parameter: *Höhe.*) Der Betrag des Schadens bzw. der Heilung wird durch die Schadensberechnung bestimmt, siehe Abschnitt A.2.8.

Relative Stärke (Parameter: *Prozentualer Anteil*). Der Betrag des Schadens bzw. der Heilung ist der angegebene *prozentuale Anteil* des maximalen Gesundheitswerts des Ziels, ggf. aufgerundet.

Absolute Stärke (Parameter: *Höhe.*) Der Betrag des Schadens bzw. der Heilung ist genau die angegebene *Höhe*.

A.2.5 Zustandsbedingungen

Zustandsbedingungen können den Ablauf des Wettstreits zusätzlich beeinflussen. Bestimmte Auswirkungen von Aktionen können eine Zustandsbedingung bei einem Monster verursachen.

Klitschnass Der effektive Verteidigungswert eines klitschnassen Monsters wird um 25 Prozent verringert.

Brand Der effektive Angriffswert eines brennenden Monsters wird um 25 Prozent verringert. Zusätzlich erleidet das Monster Schaden in Höhe von 10 Prozent seines maximalen Gesundheitswerts als zusätzliche letzte Auswirkung jeder Aktion, die es ausführt. Dies beinhaltet insbesondere die Fälle, dass das Monster passt oder seine Aktion fehlschlägt. Schutz kann ein Monster *nicht* vor Brandschaden bewahren.

Treibsand Die Geschwindigkeitswert eines Monsters ist um 25 Prozent verringert, wenn es in Treibsand gefangen ist.

Schlaf Das Monster kann keine Aktionen ausführen.

In jeder Runde besteht die Chance, dass eine Zustandsbedingung endet, siehe Abschnitt A.2.7.

A.2.6 Element

Monster sowie Aktionen gehören je einem Element an. Es gibt genau die Elemente *Normal*, *Wasser*, *Feuer*, und *Erde*. Für je zwei Elemente X, Y ist festgelegt, ob X gegen Y sehr effektiv, normal effektiv oder nicht sehr effektiv ist:

$$\mathit{istSehrEffektivGegen}(X, Y) :\Leftrightarrow (X, Y) \in \{(\text{Wasser}, \text{Feuer}), (\text{Feuer}, \text{Erde}), (\text{Erde}, \text{Wasser})\}$$

$$\mathit{istNichtSehrEffektivGegen}(X, Y) :\Leftrightarrow \mathit{istSehrEffektivGegen}(Y, X)$$

$$\mathit{istNormalEffektivGegen}(X, Y) :\Leftrightarrow \text{Normal} \in \{X, Y\} \vee X = Y$$

Diese Relationen finden Verwendung in der Schadensberechnung, siehe Abschnitt A.2.8.

A.2.7 Wettstreit

In einem Wettstreit treten inital zwei oder mehr Monster gegeneinander an. Ein Wettstreit läuft in *Runden* ab, und jede Runde hat drei *Phasen*, deren Ablauf im Folgenden erläutert ist.

Phase 0: Falls weniger als zwei kampffähige Monster übrig sind, endet der Wettstreit. Das ggf. letzte verbliebene Monster gewinnt den Wettstreit[19a 19b].

Phase I: Sortiert nach ihrer Wettstreiter-Nummer geschieht für jedes Monster *monster* folgendes:

Aktion wählen Über die Befehlsschnittstelle wird für *monster* eine Aktion des Monsters bestimmt.

Phase II: Absteigend sortiert nach ihrem effektiven Geschwindigkeitswert sind die Monster am Zug. Kampfunfähige Monster werden dabei übersprungen. Ist *monster* am Zug[4], geschieht folgendes:

Aktion auswerten Die gewählte Aktion von *monster* wird ausgewertet. Die Auswirkungen der Aktionen werden dabei eine nach der anderen an die Queue der später auszuführenden Auswirkungen angehängt.
Ist die Auswirkung eine *Wiederholung*, wird die Anzahl der Wiederholungen ***n*** bestimmt und der „Schleifenrumpf“ der *Wiederholung* ***n*** mal an die Queue angehängt. Die *Wiederholung* selbst erscheint *nicht* in der Queue.

Zustand auswerten Falls *monster* an einer Zustandsbedingung leidet, endet sie mit einer Wahrscheinlichkeit von 1/3. Das Ergebnis (Zustandsbedingung endet oder besteht weiterhin) wird jetzt in einer Meldung ausgegeben[12b 12c 13b 13c 14b 14d 15b 15c]. Besteht die Zustandsbedingung weiterhin, wird nun ihre Auswirkung ausgewertet: Schläft das Monster, wird die Queue geleert. Brennt das Monster, wird eine Schadensauswirkung mit relativem Schaden in Höhe von 10 Prozent an die Queue angehängt, bei der die Ausgabe daran angepasst ist, dass der Schaden durch Brand entstanden ist[14c].

Auswirkungen ausführen Die Auswirkungen in der Queue werden der Reihe nach ausgeführt[56]. Es findet die Auswertung der Trefferrate und danach ggf. die Schadensberechnung statt[7a 7b 8].

- Trifft die erste Auswirkung in der Queue nicht, schlägt die ganze Aktion fehl[9] und die weiteren Auswirkungen werden nicht mehr ausgeführt.
- Trifft eine andere Auswirkung in der Queue nicht, wird ohne Meldung mit der Auswertung der nächsten Auswirkung fortgefahren.
- Sinkt der Gesundheitswert des von einer Schadensauswirkung getroffenen Monsters auf 0, ist es besiegt. Dies wird unmittelbar nach der Schadensauswirkung in einer Meldung ausgegeben[18].

Ende der Runde: Für diejenigen Monster mit Schutzauswirkungen, die nach Ablauf der aktuellen Runde enden sollen, wird in der Reihenfolge der Wettstreiter-Nummer die entsprechende Meldung ausgegeben[17e].

A.2.8 Schadensberechnung

Verursacht eine Aktion eines Monsters *user* auf ein Monster *target* einen *Basisschaden* mit dem Wert *baseValue*, wird der resultierende Schadenswert als Produkt sämtlicher nachfolgend aufgelisteter Faktoren berechnet. Die abgekürzten Bezeichnungen der Zustandswerte bedeuten den *effektiven* Zustandswert (siehe Abschnitt A.2.3) des jeweils anwendenden oder getroffenen Monsters.

Beachten Sie ggf. die zu den Faktoren gehörigen Ausgabemeldungen, die unmittelbar nach der Berechnung des jeweiligen Faktors ausgegeben werden sollen.

Beachten Sie außerdem, dass sich der Gleiches-Element-Faktor ausschließlich auf das ausführende Monster und die ausgeführte Aktion bezieht, wohingegen der Elementsfaktor sich auf das ausführende und das getroffene Monster bezieht.

Basisschaden Der Wert des Basisschadens, wie in der Deklaration der Aktion angegeben.

Elementsfaktor Entspricht 2, falls das Element der Aktion sehr effektiv (siehe A.2.6) gegen das Element von *target* ist, 0.5, falls es nicht sehr effektiv ist, und sonst 1. Dafür wird im Rahmen der Ausführung einer Aktion *nur* bei der erstmaligen Schadensberechnung eine Meldung ausgegeben[7a][7b].

Zustandsfaktor Entspricht $\frac{ATK_{user}}{DEF_{target}}$.

Volltreffer-Faktor Entspricht 2 mit einer Wahrscheinlichkeit von $10^{\frac{-SPD_{\text{target}}}{SPD_{\text{user}}}} * 100$ Prozent (in dem Fall wird eine entsprechende Meldung ausgegeben[8]), sonst 1. Der Zufallsgenerator soll auch dann verwendet werden, wenn die Wahrscheinlichkeit 100 Prozent beträgt oder übersteigt.

Gleiches-Element-Faktor Entspricht 1.5, falls das Element der Aktion mit dem Element von *user* übereinstimmt, sonst 1.

Zufallsfaktor Eine zufällig generierte Zahl zwischen 0.85 und 1.

Normalisierungsfaktor Entspricht $\frac{1}{3}$.

Die Berechnung wird durchgängig `double`-wertig durchgeführt. Das Gesamtprodukt wird gegebenenfalls zur nächsten ganzen Zahl aufgerundet.

Beispiel: `BullFrog` (Element `WATER`, `ATK 78`) setzt `TidalSlam` (Element `WATER`, Basisschaden 70) gegen `FireSnail` (Element `FIRE`, `DEF 92`) ein.

$$totalDamage = \left\lceil 70 * 2 * \frac{78}{92} * 1 * 1.5 * 0.96531 * \frac{1}{3} \right\rceil = \lceil 57.289 \rceil = 58$$

A.2.9 Trefferrate

Für jede Auswirkung einer Aktion, die eine Trefferrate *hitRate* hat, bestimmt der Zufallszahlengenerator, ob sie trifft. Das Ergebnis wird in zwei Schritten berechnet:

1. Wenn das ausführende Monster oder das Ziel-Monster kampfunfähig ist, schlägt die Auswirkung fehl, ohne dass der Zufallszahlengenerator verwendet wird.

2. Die Auswirkung trifft mit einer Wahrscheinlichkeit, die sich aus dem Produkt folgender Faktoren zusammensetzt:

   Basistrefferrate Die Trefferrate der Auswirkung in Prozent, wie in der Deklaration der Aktion angegeben.

   Zustandsquotient Entspricht $\frac{PRC_{user}}{AGL_{target}}$, falls die Auswirkung ein gegnerisches Monster trifft, sonst PRC_{user}.

   Das Produkt wird nicht gerundet.

Zufallsbehaftete Auswirkungen, die ein gegnerisches Monster treffen können, sind: Schaden, Zustandsbedingung, Zustandsveränderung, Heilung.

A.2.10 Zufallsgenerator

Um sicherzustellen, dass Programmabläufe zuverlässig wiederholbar sind, muss der Zufallsgenerator (d.h. das `Random`-Objekt) entsprechend konfiguriert und bedient werden. Beachten Sie deshalb folgende Anmerkungen:

- Nutzen Sie den optionalen Kommandozeilenparameter `seed`, um Ihren Zufallsgenerator `random` zu Beginn einer Programmausführung (nicht zu Beginn eines Wettstreits) in einen deterministischen inneren Zustand zu versetzen. Wurde stattdessen der Debug-Modus aktiviert, fragen Sie das Ergebnis jeweils durch die **unten angegebene Abfrage** ab, die Sie vor der Eingabe ausgeben. Anstelle von `<context>` können Sie jeweils eine eigene Zeichenfolge angeben, die in englischer Sprache beschreibt, worüber entschieden wird, wie etwa `critical hit` oder `sleep end` und keinen Doppelpunkt oder Zeilenumbrüche enthält.
- Um mit einer Wahrscheinlichkeit von a `true` zu generieren ($a \in [0, 100]$) und sonst `false`, nutzen Sie den Ausdruck `random.nextDouble()*100 <= a`.

  Dies wird verwendet für: Volltreffer, Trefferrate, Ende einer Zustandsbedingung.

  Debug-Modus[20a]: `Decide <context>: yes or no (y/n)?`
- Um eine gleichverteilte, reelle Zufallszahl im Intervall $[a, b)$ zu generieren, nutzen Sie den Ausdruck `random.nextDouble(a, b)`.

  Dies wird verwendet für: Zufallsfaktor der Schadensberechnung.

  Debug-Modus[20b]: `Decide <context>: a double between` a `and` b`?`
- Um eine gleichverteilte, ganze Zufallszahl im Intervall $[m, n]$ zu generieren ($m, n \in \mathbb{N}$), nutzen Sie den Ausdruck `random.nextInt(m, n + 1)`.

  Dies wird verwendet für: zufällige Anzahl Wiederholungen; zufällige Anzahl Runden, die Schutz andauert

  Debug-Modus[20c]: `Decide <context>: an integer between` m `and` n`?`
- Für eine Auswirkung, die fehlschlägt, sollen mögliche weitere Zufallszahlen nicht mehr berechnet werden.
- Werten Sie die zufallsbehafteten Faktoren der Schadensberechnung in der Reihenfolge aus, in der sie in Abschnitt A.2.8 angegeben sind.

A.3 Hinweise zur Implementierung

Nach dem Start nimmt Ihr Programm über die Kommandozeile mittels der Standardeingabe Eingaben entgegen, die im Folgenden näher spezifiziert werden. Alle Befehle werden auf dem aktuellen Zustand des Programms ausgeführt. Nach Abarbeitung einer Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Im Folgenden werden Eingabezeilen mit einer schließenden spitzen Klammer (`>`) gefolgt von einem Leerzeichen eingeleitet. Diese beiden Zeichen sind kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung von Ein- und Ausgabezeilen.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die gegebenen semantischen und syntaktischen Spezifikationen nicht verletzt werden und geben Sie bei Verletzung der Spezifikationen immer eine aussagekräftige Fehlermeldung aus. Wenn die Benutzereingabe nicht dem vorgegebenen Format entspricht, ist auch eine Fehlermeldung auszugeben. Nach der Ausgabe einer Fehlermeldung soll das Programm wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Die Fehlermeldung sollte so geformt sein, dass für den Benutzer erkenntlich ist, warum eine Eingabe abgelehnt wurde.

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Setzen Sie nur die in der Aufgabenstellung angegebenen Informationen um. Geben Sie auch keine zusätzlichen Informationen aus. Bei Fehlermeldungen dürfen Sie den englischsprachigen Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error,` beginnen und darf keine Sonderzeichen, wie beispielsweise Zeilenumbrüche oder Umlaute, enthalten.

Wenn nicht anders angegeben, ist für Eingabe immer die Standardeingabe `System.in` zu verwenden. Wenn nicht anders angegeben, ist für Ausgaben immer die Standardausgabe `System.out` zu verwenden. Für Fehlermeldungen kann anstelle der Standardausgabe optional die Standardfehlerausgabe `System.err` verwendet werden. Weisen Sie diese Standardeingabe und -ausgabe niemals neu zu.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen für Platzhalter stehen, die bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern. Vergleichen Sie hierzu auch den Beispielablauf.

A.4 Funktionalität

Im Folgenden wird die erforderliche Funktionalität des Systems beschrieben.

Beachten Sie: Die Ausgabe des Programmgeschehens wird zu großen Teilen durch die Meldungen in Abschnitt A.6 abgedeckt. Die Ausgaben sollen sich an die Reihenfolge halten, wie sie in dieser Aufgabenstellung und in den Beispielen beschrieben ist. Sie finden an vielen Stellen in diesem Dokument, wo eine entsprechende Meldung ausgegeben werden soll, deren Nummern als hochgestellten Verweis.

A.4.1 Programmstart

Über die Kommandozeile wird der Pfad zu einer Konfigurationsdatei sowie optional ein Startwert (*seed*) für den Zufallsgenerator oder das Flag `debug` übermittelt. Aus der Konfigurationsdatei werden anschließend die Aktionen und Monster eingelesen. ~~Die eingelesenen Monster erhalten dabei ihre *Wettstreiter-Nummer* in der Reihenfolge, in der sie in der Konfigurationsdatei deklariert werden, beginnend bei 1.~~ Das Flag `debug` soll den Debug-Modus Ihres Zufallsgenerators aktivieren, in dem jede Entscheidung über die Kommandozeile getroffen werden kann, siehe Abschnitt A.2.10.

Schließlich wird die gesamte Konfiguration *verbatim* ausgegeben. Ist die Konfiguration ungültig, wird das Programm mit einer Fehlermeldung beendet. Andernfalls folgt eine Meldung mit der Anzahl der eingelesenen Aktionen und Monster[1]. Die Syntax einer Konfigurationsdatei wird in Abschnitt A.5 erläutert.

Eingabe `java␣Main␣<path>␣[<seed>|debug]`

Ausgabe

```
<config>

Loaded <n> actions, <m> monsters.
> [...]
```

Beispielinteraktion

```
> java -jar MonsterBattle.jar /home/myname/documents/config.txt 55378
attack SingLullaby NORMAL inflictStatusCondition SLEEP 70 end attack
[...]
monster FireSnail FIRE 438 54 92 23 VolcanoStorm

Loaded 9 actions, 3 monsters.
> [...]
```

A.4.2 Debug-Eingaben

Sollte das Programm im Debug-Modus laufen, können in Phase II einer Wettstreit-Runde Debug-Eingabeaufforderungen auftreten. In diesem Fall sind neben der geforderten Angabe ausschließlich die Befehle `quit` und `show` zugelassen. Ist die Nutzereingabe ungültig, wird nach der entsprechenden Fehlermeldung jeweils die Meldung zur Eingabeaufforderung wiederholt.

Beispielinteraktion

```
It's Dog's turn!

Decide attack hit: yes or no? (y/n)
> yes
Error, enter y or n.

Decide attack hit: yes or no? (y/n)
> show monsters
Error, enter y or n.

Decide attack hit: yes or no? (y/n)
> y
[...]

Decide repeat count: an integer between 2 and 5?
> 7
Error, out of range.

Decide repeat count: an integer between 2 and 5?
> 4
```

A.4.3 Grundlegende Befehle

Im Folgenden sind die Befehle definiert, die in- und außerhalb von Wettstreiten gültig sind.

A.4.3.1 Der Befehl `quit`

Dieser Befehl beendet das Programm vollständig. Er nimmt keinerlei Argumente entgegen. Beachten Sie, dass keine Methoden wie `System.exit()` oder `Runtime.exit()` zum Beenden des Programms verwendet werden dürfen.

Eingabe `quit`

Beispielinteraktion

```
> quit
```

A.4.3.2 Der Befehl `load`

Dieser Befehl lädt eine Konfiguration aus einer Datei, die die aktuelle Konfiguration ersetzt. Ist die Konfiguration ungültig, wird die aktuelle Konfiguration beibehalten. Der ggf. laufende Wettstreit wird ohne Meldung beendet. Analog zum Programmstart wird die Konfiguration *verbatim* ausgegeben und mit einer Meldung[1] zusammengefasst.

Eingabe `load␣<path>`

Beispielinteraktion

```
> load /home/myname/documents/config2.txt
<config>

Loaded <n> actions, <m> monsters.
> [...]
```

A.4.3.3 Der Befehl `competition`

Startet einen neuen Wettstreit mit den durch ihre Namen angegebenen Monstern[2]. Der Befehl schlägt fehl, wenn eines der Monster unbekannt ist. Findet bereits ein Wettstreit statt, wird dieser verworfen und durch den neuen Wettstreit ersetzt. Die Monster erhalten dabei ihre *Wettstreiter-Nummer* in der Reihenfolge, in der sie angeführt werden, beginnend bei 1. Ist der Monstername *name* mehrfach aufgeführt, erhalten sämtliche Monster dieses Namens eine eigene fortlaufende Nummer *n* beginnend bei 1, die stets als Suffix in der Form `#n` an den Namen angehängt wird, wenn der Name des Monsters ausgegeben wird.

Eingabe `competition␣<monsterName1>␣<monsterName2>␣[...<monsterNameN>]`

Ausgabe `The␣<N>␣monsters␣enter␣the␣competition!`

Beispielinteraktion

```
> competition BullFrog FireSnail FireSnail
The 3 monsters enter the competition!

What should BullFrog do?
> [...]
```

A.4.3.4 Der Befehl `show monsters` Gibt die aus der Konfiguration bekannten Monster mit Namen, Element und den Zustandswerten HP, ATK, DEF und SPD an.

Beispielinteraktion

```
> show monsters
BullFrog: Element WATER, HP 312, ATK 78, DEF 66, SPD 52
FireSnail: Element FIRE, HP 438, ATK 54, DEF 92, SPD 23

> [...]
```

A.4.4 Zusätzliche Befehle während eines Wettstreits

Während eines Wettstreits werden in Phase I Eingaben erwartet, wo für ein Monster mit dem Namen *monster* abgefragt wird, welche Aktion *monster* ausführen soll. Vor *jeder* Nutzereingabe

soll die entsprechende Ausgabe[3] wiederholt werden. Das beinhaltet insbesondere den Fall, dass die vorige Nutzereingabe ungültig war.

Die folgenden Befehle sind während Phase I einer Runde zusätzlich verfügbar (*nicht* bei einer Debug-Abfrage):

A.4.4.1 Der Befehl `show`

Listet alle Monster auf, die am Wettstreit teilnehmen (auch, wenn sie kampfunfähig sind). Für alle Monster wird in der Reihenfolge ihrer Wettstreiter-Nummer jeweils folgendes ausgegeben:

- Der aktuelle Gesundheitswert, dargestellt durch das Wort `[X`^{m}`_`^{n}`]`, wobei $m = \left\lceil \frac{20 * HP_{current}}{HP_{max}} \right\rceil$ und $n = 20 - m$.
- Nummer und Name des Monsters. Für das Monster, dessen Aktion gerade gewählt werden soll, wird dem Namen ein Stern `*` vorangestellt.
- Die Bezeichnung für die aktuelle Zustandsbedingung (`WET`, `BURN`, `QUICKSAND`, `SLEEP`) bzw. `FAINTED` für kampfunfähige Monster und `OK` für Monster ohne Zustandsbedingung.

Eingabe `show`

Beispielinteraktion

```
What should BullFrog do?
> show
[XXXXXXXXXXXXXX____] 1 *BullFrog (BURN)
[XXXXXXXXXX________] 2 FireSnail#1 (OK)
[__________________] 3 FireSnail#2 (FAINTED)

What should BullFrog do?
> [...]
```

A.4.4.2 Der Befehl `show actions`

Listet die Aktionen des Monsters auf, dessen Aktion gewählt werden soll. Die Reihenfolge entspricht dabei der Reihenfolge, in der die Aktionen in der Deklaration des Monsters angegeben sind. Für jede Aktion wird je einer Zeile ihr Name, ihr Element, die Art (als Präfix `'b'`, `'r'` respektive `'a'` für Basis-, relativen und absoluten Schaden) und Höhe/Anteil der ersten Schadensauswirkung, sowie die Trefferrate der ersten Auswirkung angegeben. Falls die Aktion keine Schadensauswirkung beinhaltet, wird stattdessen die Zeichenfolge `'--'` ausgegeben. Ist die erste Auswirkung eine *Wiederholung*, wird die Trefferrate der ersten Auswirkung im Schleifenrumpf ausgegeben.

Eingabe `show␣actions`

Ausgabe

```
ACTIONS OF <monster_name>
<action1_name>: ELEMENT <element>, Damage <damage>, HitRate <hitRate>
[...]
```

Beispielinteraktion

```
What should BullFrog do?
> show actions
ACTIONS OF BullFrog
TidalSlam: ELEMENT WATER, Damage b70, HitRate 90
StickyTongue: ELEMENT NORMAL, Damage --, HitRate 85
PondLeap: ELEMENT WATER, Damage b50, HitRate 95

What should BullFrog do?
>
```

A.4.4.3 Der Befehl `action`

Bestimmt die nächste Aktion des Monsters, dessen Aktion gewählt werden soll. Falls die Aktion ein gegnerisches Ziel benötigt *und* mehr als ein gegnerisches Monster vorhanden sind, wird der Name eines Monsters obligatorisch angegeben; falls nur noch ein gegnerisches Monster verbleibt, muss dessen Name nicht angegeben werden. Falls das aktuelle Monster keine Aktion mit dem angegebenen Namen beherrscht oder das angegebene Monster unbekannt ist, schlägt der Befehl fehl und die Abfrage wird mit demselben Monster fortgesetzt.

Eingabe `action␣<actionName>␣[<monsterName>]`

Beispielinteraktion

```
What should FireSnail do?
> action LavaDrip BullFrog

What should Dog do?
> [...]
```

A.4.4.4 Der Befehl `pass`

Bestimmt, dass das Monster, dessen Aktion gewählt werden soll, in dieser Runde passt.

Eingabe `pass`

Beispielinteraktion

```
What should BullFrog do?
> pass

What should FireSnail do?
> [...]
```

A.4.4.5 Der Befehl `show stats`

Gibt den aktuellen Zustand des Monsters aus, dessen Aktion gewählt werden soll. Für jeden Zustandswert wird dabei dessen Abkürzung und der entsprechende Wert sowie ggf. zusätzliche Informationen in einer gemeinsamen Zeile ausgegeben, nämlich für den Gesundheitswert `'/'` gefolgt

vom maximalen Gesundheitswert, sowie für die anderen Zustandswerte die Zustandsänderung in Klammern, sofern sie von 0 verschieden ist.

Eingabe `show␣stats`

Ausgabe

```
STATS OF <monster_name>
HP <hp>/<max_hp>, ATK <atk>[(<atk_change>)], DEF <def>[(<def_change>)], SPD
    <spd>[(<spd_change>)], PRC 1[(<prc_change>)], AGL 1[(<agl_change>)]
```

Beispielinteraktion

```
What should BullFrog do?
> show stats
STATS OF BullFrog
HP 250/312, ATK 78, DEF 66(-1), SPD 52, PRC 1(+3), AGL 1

What should BullFrog do?
> [...]
```

A.5 Format einer Konfiguration

Eine Konfigurationsdatei deklariert Aktionen und Monster. Deklarationen von Aktionen enthalten dabei insbesondere eine Auflistung ihrer Auswirkungen. Angaben von Auswirkungen beginnen stets mit einem Schlüsselwort gefolgt von Argumenten~~, die durch ein Semikolon abgeschlossen werden~~. Deklarationen von Monstern bestehen aus ihrem Namen, ihrem Element, und einer Auflistung ihres Grundzustandes. Die Namen von Monstern und Aktionen sollen jeweils paarweise verschieden sein.

Die gültige Struktur von Konfigurationsdateien ist unten als Grammatik einer formalen Sprache angegeben. Einige verwendete Syntaxelemente sind im Folgenden erläutert:

Alternative („x oder y“)	`x | y`
m- bis n-fache Wiederholung	`x{m,n}`
Beliebige Wiederholung (kurz für `x{0,∞}`)	`x*`
Mind. eine Wiederholung (kurz für `x{1,∞}`)	`x+`
Alphanumerische Zeichenfolge ohne Leerzeichen	`<action_name>`, `<monster_name>`
Nichtnegative Ganzzahl	`<value>`, `<min>`, `<max>`
Positive Ganzzahl	`<max_health>`, `<base_attack>`, `<base_defense>`, `<base_speed>`
Ganzzahl im Intervall $[0, 100]$	`<hit_rate>`, `<percentage>`
Ganzzahl mit optionalem Vorzeichen (+,-)	`<change>`
Systemunabhängiger Zeilenumbruch	`<nl>`

➞ Grammatik

```
CONFIG ::= ACTION* MONSTER*

ACTION ::= 'action' <action_name> ELEMENT <nl> EFFECTS 'end action' <nl>+

EFFECTS ::= (EFFECT <nl>)+

EFFECT ::= 'damage' TARGET_MONSTER STRENGTH <hit_rate>
    | 'inflictStatusCondition' TARGET_MONSTER STATUS_CONDITION <hit_rate>
    | 'inflictStatChange' TARGET_MONSTER STAT <change> <hit_rate>
    | 'protectStat' PROTECT_TARGET COUNT <hit_rate>
    | 'heal' TARGET_MONSTER STRENGTH <hit_rate>
    | 'repeat' COUNT <nl> EFFECTS 'end repeat'
    | 'continue' <hit_rate>

STRENGTH ::= 'base' <value>
    | 'rel' <percentage>
    | 'abs' <value>

MONSTER ::= 'monster' <monster_name> ELEMENT <max_health>
   <base_attack> <base_defense> <base_speed>
   <action_name>{1,4} <nl>+

COUNT ::= <value> |'random' <min> <max>
ELEMENT ::= 'WATER' |'FIRE' | 'EARTH' | 'NORMAL'
STATUS_CONDITION ::= 'WET' |'BURN' | 'QUICKSAND' | 'SLEEP'
STAT ::= 'ATK' | 'DEF' | 'SPD' | 'PRC' | 'AGL'
PROTECT_TARGET ::= 'health' |'stats'
TARGET_MONSTER ::= 'user' |'target'
```

Hinweise zum Parsen der Konfiguration:

- Die Grammatik gibt durch die Zeilenumbrüche `<nl>` klar vor, welchen Inhalt eine Zeile haben kann. Sie können also stets eine Zeile als Ganzes parsen und erwarten, dass bestimmte Angaben in entsprechender Anzahl enthalten sind.
- Nach jedem Zeilenumbruch sind eine oder mehrere Leerzeilen zur Strukturierung erlaubt, die bei der Verarbeitung ignoriert werden sollen.
- Zwischen den Argumenten wird je genau ein *Whitespace*-Zeichen erwartet; am Anfang jeder Zeile sind beliebig viele Leerzeichen oder Tabulatoren erlaubt.
- Abweichend von der Grammatikdefinition ist es *nicht* erforderlich, dass eine Konfigurationsdatei mit einer Leerzeile endet.
- Wir legen Ihnen nahe, zum Parsen der Konfigurationsdatei die Methode `Files#readAllLines(Path)` zu verwenden.

A.6 Ausgabe

Hier finden Sie eine Liste einiger Ausgaben, die im Programmablauf entstehen können. Vor denjenigen Ausgaben, die mit einem Stern * markiert sind, soll zusätzlich eine Leerzeile ausgegeben werden, um die Übersichtlichkeit zu verbessern. Sie finden in Kapitel 3 und 4 Verweise auf diejenigen Ausgabenummern, die an der jeweiligen Stelle relevant sind.

```
 1  *  Loaded <number> actions, <number> monsters.
 2     The <number> monsters enter the competition!
 3  *  What should <monster> do?
 4  *  It's <monster>'s turn.
 5     <monster> passes!
 6     <monster> uses <action>!
7a     It is very effective!
7b     It is not very effective...
 8     Critical hit!
 9     The action failed...
10     <monster> takes <amount> damage!
11     <monster> gains back <amount> health!
12a    <monster> falls asleep!
12b    <monster> is asleep!
12c    <monster> woke up!
13a    <monster> becomes soaking wet!
13b    <monster> is soaking wet!
13c    <monster> dried up!
14a    <monster> caught on fire!
14b    <monster> is burning!
14c    <monster> takes <amount> damage from burning!
14d    <monster>'s burning has faded!
15a    <monster> gets caught by quicksand!
15b    <monster> is caught in quicksand!
15c    <monster> escaped the quicksand!
16a    <monster>'s <stat> rises!
16b    <monster>'s <stat> decreases...
17a    <monster> is now protected against damage!
17b    <monster> is now protected against status changes!
17c    <monster> is protected and takes no damage!
17d    <monster> is protected and is unaffected!
17e    <monster>'s protection fades away...
18     <monster> faints!
19a  *  <monster> has no opponents left and wins the competition!
19b  *  All monsters have fainted. The competition ends without a winner!
20a  *  Decide <context>: yes or no? (y/n)
20b  *  Decide <context>: a double between <a> and <b>?
20c  *  Decide <context>: an integer between <m> and <n>?
```

A.7 Beispiele

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung in den gegebenen Beispielen. Eingabezeilen der Beispielinteraktion werden mit der Zeichenkette `'>␣'` eingeleitet; diese ist ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dient nur der Kennzeichnung von Eingabezeilen.

Im ILIAS-Ordner finden Sie die Beispiele als Textdateien, sowie eine Datei, in der die Auswertungen des Zufallsgenerators im untenstehenden Beispiel dargestellt sind.

A.7.1 Konfiguration

Beispiel

```
1  action TidalSlam WATER
2      damage target base 70 90
3  end action
4
5  action StickyTongue NORMAL
6      inflictStatChange target SPD -1 85
7  end action
8
9  action CroakCall NORMAL
10     protectStat stats random 1 3 100
11 end action
12
13 action PondLeap WATER
14     damage target base 50 95
15 end action
16
17 action BlazingShell FIRE
18     damage target base 40 100
19 end action
20
21 action LavaDrip FIRE
22     damage target base 30 80
23 end action
24
25 action MagmaTrail FIRE
26     inflictStatusCondition target BURN 90
27 end action
28
29 action InfernoBurst FIRE
30     damage target base 100 75
31 end action
32
33 action Snap NORMAL
34     damage target abs 5 95
```

```
35 end action
36
37 action Devour EARTH
38     damage target base 10 80
39     heal user abs 15 100
40 end action
41
42 action Howl NORMAL
43     inflictStatChange target PRC -1 85
44 end action
45
46 monster BullFrog WATER 112 78 66 32 TidalSlam StickyTongue CroakCall
       PondLeap
47 monster FireSnail FIRE 238 54 92 23 BlazingShell LavaDrip MagmaTrail
       InfernoBurst
48 monster VenusFlytrap EARTH 294 81 74 41 Snap Devour
49 monster Dog NORMAL 76 72 58 65 Snap Howl
```

A.7.2 Interaktion

Beispielinteraktion

```
1  > java -jar MonsterCompetition.jar ./config.txt 22
2  action TidalSlam WATER
3      damage target base 70 90
4  end action
5
6  action StickyTongue NORMAL
7      inflictStatChange target SPD -1 85
8  end action
9
10 action CroakCall NORMAL
11     protectStat stats random 1 3 100
12 end action
13
14 action PondLeap WATER
15     damage target base 50 95
16 end action
17
18 action BlazingShell FIRE
19     damage target base 40 100
20 end action
21
22 action LavaDrip FIRE
23     damage target base 30 80
24 end action
```

```
25
26  action MagmaTrail FIRE
27      inflictStatusCondition target BURN 90
28  end action
29
30  action InfernoBurst FIRE
31      damage target base 100 75
32  end action
33
34  action Snap NORMAL
35      damage target abs 5 95
36  end action
37
38  action Devour EARTH
39      damage target base 10 80
40      heal user abs 15 100
41  end action
42
43  action Howl NORMAL
44      inflictStatChange target PRC -1 85
45  end action
46
47  monster BullFrog WATER 112 78 66 32 TidalSlam StickyTongue CroakCall
        PondLeap
48  monster FireSnail FIRE 238 54 92 23 BlazingShell LavaDrip MagmaTrail
        InfernoBurst
49  monster VenusFlytrap EARTH 294 81 74 41 Snap Devour
50  monster Dog NORMAL 76 72 58 65 Snap Howl
51
52  Loaded 11 actions, 4 monsters.
53  > show monsters
54  BullFrog: ELEMENT WATER, HP 112, ATK 78, DEF 66, SPD 32
55  FireSnail: ELEMENT FIRE, HP 238, ATK 54, DEF 92, SPD 23
56  VenusFlytrap: ELEMENT EARTH, HP 294, ATK 81, DEF 74, SPD 41
57  Dog: ELEMENT NORMAL, HP 76, ATK 72, DEF 58, SPD 65
58  > competition BullFrog FireSnail
59  The 2 monsters enter the competition!
60
61  What should BullFrog do?
62  > show actions
63  ACTIONS OF BullFrog
64  TidalSlam: ELEMENT WATER, Damage b70, HitRate 90
65  StickyTongue: ELEMENT NORMAL, Damage --, HitRate 85
66  CroakCall: ELEMENT NORMAL, Damage --, HitRate 100
67  PondLeap: ELEMENT WATER, Damage b50, HitRate 95
68
69  What should BullFrog do?
70  > action PondLeap
71
72  What should FireSnail do?
```

```
> show actions
ACTIONS OF FireSnail
BlazingShell: ELEMENT FIRE, Damage b40, HitRate 100
LavaDrip: ELEMENT FIRE, Damage b30, HitRate 80
MagmaTrail: ELEMENT FIRE, Damage --, HitRate 90
InfernoBurst: ELEMENT FIRE, Damage b100, HitRate 75

What should FireSnail do?
> action MagmaTrail

It's BullFrog's turn.
BullFrog uses PondLeap!
It is very effective!
FireSnail takes 41 damage!

It's FireSnail's turn.
FireSnail uses MagmaTrail!
BullFrog caught on fire!

What should BullFrog do?
> action TidalSlam

What should FireSnail do?
> action InfernoBorst
Error, FireSnail does not know the action InfernoBorst.

What should FireSnail do?
> action InfernoBurst

It's BullFrog's turn.
BullFrog is burning!
BullFrog uses TidalSlam!
The action failed...
BullFrog takes 12 damage from burning!

It's FireSnail's turn.
FireSnail uses InfernoBurst!
It is not very effective...
BullFrog takes 21 damage!

What should BullFrog do?
> action TidalSlam

What should FireSnail do?
> action InfernoBurst

It's BullFrog's turn.
BullFrog is burning!
```

```
121 BullFrog uses TidalSlam!
122 It is very effective!
123 Critical hit!
124 FireSnail takes 77 damage!
125 BullFrog takes 12 damage from burning!
126
127 It's FireSnail's turn.
128 FireSnail uses InfernoBurst!
129 It is not very effective...
130 BullFrog takes 20 damage!
131
132 What should BullFrog do?
133 > show
134 [XXXXXXXXX__________] 1 *BullFrog (BURN)
135 [XXXXXXXXXXX________] 2 FireSnail (OK)
136
137 What should BullFrog do?
138 > show stats
139 STATS OF BullFrog
140 HP 47/112, ATK 78, DEF 66, SPD 32, PRC 1, AGL 1
141
142 What should BullFrog do?
143 > action PondLeap
144
145 What should FireSnail do?
146 > action InfernoBurst
147
148 It's BullFrog's turn.
149 BullFrog's burning has faded!
150 BullFrog uses PondLeap!
151 It is very effective!
152 FireSnail takes 41 damage!
153
154 It's FireSnail's turn.
155 FireSnail uses InfernoBurst!
156 It is not very effective...
157 Critical hit!
158 BullFrog takes 36 damage!
159
160 What should BullFrog do?
161 > action PondLeap
162
163 What should FireSnail do?
164 > action MagmaTrail
165
166 It's BullFrog's turn.
167 BullFrog uses PondLeap!
168 It is very effective!
```

```
169 FireSnail takes 40 damage!
170
171 It's FireSnail's turn.
172 FireSnail uses MagmaTrail!
173 BullFrog caught on fire!
174
175 What should BullFrog do?
176 > pass
177
178 What should FireSnail do?
179 > action InfernoBurst
180
181 It's BullFrog's turn.
182 BullFrog is burning!
183 BullFrog passes!
184 BullFrog takes 12 damage from burning!
185 BullFrog faints!
186
187 It's FireSnail's turn.
188 FireSnail uses InfernoBurst!
189 The action failed...
190
191 FireSnail has no opponents left and wins the competition!
192 > quit
```