Karlsruher Institut für Technologie

PD Dr. rer. nat. Robert Heinrich
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

Programmieren – Wintersemester 2024

Abschlussaufgabe 1
Version 1.0

20 Punkte

Ausgabe:	14.02.2024, ca. 12:00 Uhr
Abgabestart:	28.02.2024, 12:00 Uhr
Abgabefrist:	14.03.2024, 06:00 Uhr

Geschlechtergerechte Sprache

Wenn das generische Maskulinum gewählt wurde, geschieht dies zur besseren Lesbarkeit und zum einfachen Verständnis der Aufgabenstellung. Sofern nicht anders angegeben, beziehen sich Angaben im Sinne der Gleichbehandlung auf Vertretende aller Geschlechter.

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.

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.

Kommunikation und aktuelle Informationen

In unseren *FAQs*[1] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie

[1] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ

Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im ILIAS-Wiki[2].

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[3] 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`.
- 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.

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 ILIAS-Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.

[2] `https://ilias.studium.kit.edu/goto.php?target=wiki_2213632_Hauptseite`
[3] `https://artemis.praktomat.cs.kit.edu/`

- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im ILIAS-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 im ILIAS beschreibt, wie Checkstyle verwendet werden kann.

Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 28.02.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 14.03.2024, 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: Code Fight

In dieser Abschlussaufgabe sollen Sie *Code Fight*, ein rundenbasiertes Spiel für mindestens zwei KIs, entwickeln. Das Spiel ist inspiriert von dem Spieleklassiker *Core War*[4]. Zusammengefasst spielen zwei oder mehrere künstliche Intelligenzen (KIs) in einem gemeinsamen Speicher gegeneinander. Ziel der KIs ist es, die anderen KIs zum Absturz zu bringen. Dazu können die KIs den gemeinsamen Speicher lesen und schreiben. Die letzte KI, die noch läuft, gewinnt das Spiel.

A.1 Das Spiel im Detail

In diesem Abschnitt werden die Details des Spiels beschrieben. Das Spiel simuliert die Ausführung mehrerer Programme (KIs) in einem gemeinsamen Speicher. Jedes Programm besteht initial aus einer Folge von KI-Befehlen, die nacheinander ausgeführt werden. Ziel der KIs ist es, den Speicher so zu verändern, dass die anderen KIs nicht mehr ausgeführt werden können. Dazu kann eine KI einen speziellen KI-Befehl in den Speicher schreiben, der die Ausführung von KIs stoppt. Damit die KIs ein Verhalten haben, gibt es verschiedene KI-Befehle, die die KIs ausführen können. Diese können z.B. arithmetische Operationen oder bedingte Sprünge sein. Durch diese KI-Befehle wird somit das Verhalten der KIs bestimmt.

Das Spiel selbst startet mit dem Laden der KIs in den Speicher. Hierzu werden die definierten KI-Befehle in den gemeinsamen Speicher platziert. Anschließend wird nacheinander immer ein KI-Befehl einer KI ausgeführt. Sollte eine KI den speziellen KI-Befehl *STOP* ausführen, so wird die KI gestoppt und die Ausführung der KI wird nicht fortgesetzt.

A.2 Konzepte

In diesem Abschnitt werden die notwendigen Konzepte für das Spiel vorgestellt.

A.2.1 Speicher

Der *Speicher* ist das Spielfeld der KIs. Er besteht aus einer endlichen Anzahl an *Speicherzellen*. Wichtig ist hierbei, dass der Speicher zyklisch ist. Das bedeutet, dass die Speicherzelle nach der letzten Speicherzelle wieder die erste Speicherzelle ist. Die Spielzellen sind durchnummeriert, beginnend bei 0. Eine Speicherzelle enthält immer drei Einträge:

- **KI-Befehl:** Der KI-Befehl, der in der Speicherzelle gespeichert ist.
- **Eintrag A:** Das erste Argument des KI-Befehls.
- **Eintrag B:** Das zweite Argument des KI-Befehls.

Die KI-Befehle sind in A.2.3 erklärt. Die Argumente sind immer ganze Zahlen im Bereich von $[INT\_MIN, INT\_MAX]$. Beim Ausführen von KI-Befehlen sind Überläufe oder Unterläufe explizit erlaubt und gewollt. Wird also beispielsweise der Wert 1 auf die größte darstellbare Zahl addiert, so ergibt sich das Ergebnis INT_MIN.

[4] `https://de.wikipedia.org/wiki/Core_War`

A.2.2 KIs: Die Programme des Spiels

Die KIs sind die Programme, die im Speicher ausgeführt werden. Ein Programm ist im Spiel eine Folge von KI-Befehlen mit Argumenten. Ein Programm besteht hierbei aus mindestens einem KI-Befehl. Entscheidend für die Ausführung eines Programms ist die Speicherzelle, die das Programm als nächstes Ausführt. Im Normalfall ist dies die Speicherzelle direkt nach dem zuletzt ausgeführten KI-Befehl. Es gibt aber auch KI-Befehle, die die Ausführung an eine andere Speicherzelle springen lassen.

A.2.3 KI-Befehle

In der folgenden Tabelle A.1 werden die verschiedenen KI-Befehle der KIs kurz beschrieben. Außerdem enthält dieser Abschnitt noch eine detaillierte Beschreibung aller KI-Befehle.

A.2.3.1 KI-Befehl: STOP Für diesen KI-Befehl sind die Argumente irrelevant. Der KI-Befehl stoppt die Ausführung der KI und die KI hat nicht gewonnen.

A.2.3.2 KI-Befehl: MOV_R Der relative move KI-Befehl (MOV_R) kopiert den Inhalt der Speicherzelle *Source* in die Speicherzelle *Target*. Die Argumente *Source* und *Target* geben hierbei relativ zum aktuellen KI-Befehl die Speicherzellen an. Würde sich beispielsweise der aktuelle KI-Befehl in Speicherzelle 5 befinden, so würde der KI-Befehl *MOV_R -3 1* den Inhalt der Speicherzelle 2 in die Speicherzelle 6 kopieren. Es würde also der als nächstes auszuführende KI-Befehl mitsamt Argumenten überschrieben werden.

A.2.3.3 KI-Befehl: MOV_I Der indirekte move KI-Befehl (MOV_I) kopiert wie schon der relative move KI-Befehl den Inhalt der Speicherzelle *Source* in die Speicherzelle *Target*. Im Unterschied zu dem relativen move KI-Befehl ist hier die Position der Speicherzelle *Target* nicht direkt relativ zum aktuellen KI-Befehl. Argument B gibt bei diesem KI-Befehl an, welche Speicherzelle für die Berechnung der Position von *Target* verwendet wird. Zur Berechnung der Position wird zunächst die Speicherzelle *Intermidiate* bestimmt. Diese ergibt sich, indem vom aktuellen KI-Befehl aus, so viele Speicherzellen weitergegangen wird, wie der Wert von Argument B angibt. In der Speicherzelle B wird dann nochmals der Wert von Argument B betrachtet und entsprechend viele Speicherzellen weitergegangen.

Das folgende Beispiel soll das Prinzip verdeutlichen. Der Pfeil in Speicherzelle 2 soll den aktuellen KI-Befehl darstellen.

KI Programm

```
0 | STOP 1 3
1 | STOP 7 8
2 | -> MOV_I 0 -2
3 | STOP 13 37
```

Zunächst wird die Speicherzelle *Source* bestimmt. Diese ergibt sich, indem vom aktuellen KI-Befehl aus, so viele Speicherzellen weitergegangen wird, wie der Wert von Argument A angibt. In diesem Fall ist das die Speicherzelle 2, also die Speicherzelle des aktuellen KI-Befehls. Als Nächstes wird die Speicherzelle *Intermidiate* bestimmt. Diese ergibt sich, indem vom aktuellen KI-Befehl aus, so viele Speicherzellen weitergegangen wird, wie der Wert von Argument B angibt. In diesem Fall ist das die Speicherzelle 0. Um die Speicherzelle *Target* zu bestimmen, wird nun der Wert von Argument B in der Speicherzelle *Intermidiate* betrachtet. In diesem Fall ist dieser Wert 3. Somit ergibt sich die Speicherzelle *Target* zu Speicherzelle 3. Der Inhalt der Speicherzelle 2 wird nun in die Speicherzelle 3 kopiert.

A.2.3.4 KI-Befehl: ADD Der KI-Befehl *ADD* addiert die Werte von Argument A und Argument B und speichert das Ergebnis in Eintrag B. Der KI-Befehl kann somit genutzt werden, um einen einfachen Zähler zu implementieren.

A.2.3.5 KI-Befehl: ADD_R Der relative add KI-Befehl (*ADD_R*) ist ähnlich zum einfachen *ADD*-KI-Befehl. Nur addiert dieser KI-Befehl den Wert von Argument A mit dem Wert von Eintrag B der Speicherzelle *Target* und speichert das Ergebnis in Eintrag B der Speicherzelle *Target*. Die Position von *Target* ist relativ zum aktuellen KI-Befehl.

Das folgende Beispiel soll das Prinzip verdeutlichen. Der Pfeil in Speicherzelle 2 soll den aktuellen KI-Befehl darstellen.

KI Programm

```
0 | STOP 1 3
1 | STOP 7 8
2 | -> ADD_R 10 -2
3 | STOP 13 37
```

Zunächst wird die Speicherzelle *Target* bestimmt. Diese ergibt sich, indem vom aktuellen KI-Befehl aus, so viele Speicherzellen weitergegangen wird, wie der Wert von Argument B angibt. In diesem Fall ist das die Speicherzelle 0. Der Wert von Eintrag B der Speicherzelle 0 ist 3. Zu diesem Wert wird nun der Wert von Argument A addiert. In diesem Fall ist das der Wert 10. Somit ergibt sich der neue Wert von Eintrag B der Speicherzelle 0 zu 13.

A.2.3.6 KI-Befehl: JMP Der KI-Befehl *JMP* springt zu der Speicherzelle, die durch den Wert von Argument A gegeben ist. Die Position ist relativ zum aktuellen KI-Befehl. Wichtig ist hierbei, dass der KI-Befehl den nächsten auszuführenden KI-Befehl definiert.

A.2.3.7 KI-Befehl: JMZ Der KI-Befehl *JMZ* springt zu der Speicherzelle, die durch den Wert von Argument A gegeben ist, wenn der Wert von Eintrag B der Speicherzelle *CheckCell* 0 ist. Die Speicherzelle *CheckCell* ist relativ zum aktuellen KI-Befehl. Wie schon beim KI-Befehl *JMP* ist hierbei wichtig, dass der KI-Befehl den nächsten auszuführenden KI-Befehl definiert. Sollte der Wert von Eintrag B der Speicherzelle *CheckCell* ungleich 0 sein, wird ganz normal der nächste KI-Befehl ausgeführt.

A.2.3.8 KI-Befehl: CMP Der KI-Befehl *CMP* vergleicht den Eintrag A der Speicherzelle *First* mit dem Eintrag B der Speicherzelle *Second*. Beide Speicherzellen sind relativ zum aktuellen KI-Befehl. Sollten die Werte ungleich sein, wird der nächste KI-Befehl übersprungen. Ansonsten hat der KI-Befehl keine Auswirkungen.

A.2.3.9 KI-Befehl: SWAP Der KI-Befehl *SWAP* vertauscht den Eintrag A von *First* mit dem Eintrag B von *Second*. Beide Speicherzellen sind relativ zum aktuellen KI-Befehl.

A.2.4 Ein einfaches Programm

In diesem Abschnitt wird ein einfaches Programm vorgestellt, welches die Funktionsweise einiger KI-Befehle verdeutlichen soll.

A.2.4.1 Hello World Das einfachste Programm ist das folgende:

KI Programm

```
0 | MOV_R 0 1
```

Dieses Programm besteht aus genau einem KI-Befehl. Der KI-Befehl ist ein *MOV_R* KI-Befehl. Dieser KI-Befehl kopiert den Inhalt der Speicherzelle 0 in die Speicherzelle 1. Die Speicherzellen sind hierbei relativ zum aktuellen KI-Befehl zu verstehen. Daher wird der Inhalt der aktuellen Speicherzelle (Speicherzelle 0) in die nächste Speicherzelle (Speicherzelle 1) kopiert. Somit enthält die Speicherzelle 1 nach der Ausführung des KI-Befehls denselben KI-Befehl mitsamt Argumenten wie die Speicherzelle 0.

A.2.4.2 Sleepy Das folgende Programm ist einem existierenden Programm[5] von John Q. Smith nachempfunden und für diese Aufgabe adaptiert worden.

KI Programm

```
0 | ADD 10 -1
1 | MOV_I 2 -1
2 | JMP -2 0
3 | STOP 13 37
```

Sleepy ist ein einfaches KI-Programm, welches in regelmäßigen Abständen Zellen mit STOP-KI-Befehlen befüllt und so versucht andere KIs zu stoppen. Das Programm besteht aus vier KI-Befehlen. Der erste KI-Befehl ist ein *ADD*-KI-Befehl. Dieser dient als Zähler, um sukzessive die Entfernung der Ziel-Speicherzelle zu erhöhen. Der zweite KI-Befehl dient dazu den STOP-KI-Befehl in die Ziel-Speicherzelle zu schreiben. Hierzu kopiert der KI-Befehl den Inhalt der Speicherzelle mit dem Inhalt `STOP 13 37` (zwei Zellen weiter). Die Zielzelle wird hierbei indirekt durch den Wert von Argument B des Zählers bestimmt (Zelle vor dem aktuellen KI-Befehl). Nach dem Kopieren wird im nächsten Schritt der Sprung (JMP) ausgeführt. Dieser hat zur Folge, dass der erste KI-Befehl

[5] `http://www.koth.org/info/corewars_for_dummies/dummies.html`

des Programms als nächstes ausgeführt wird. Dieser ist wiederum der *ADD*-KI-Befehl. Der vierte KI-Befehl ist der *STOP*-KI-Befehl. Dieser kann wegen dem *JMP*-KI-Befehl nicht erreicht werden und dient als Quelle für den *MOV_I*-KI-Befehl.

Nach drei ausgeführten KI-Befehlen sieht der Speicher wie folgt aus:

KI Programm

```
0 | -> ADD 10 9
1 | MOV_I 2 -1
2 | JMP -2 0
3 | STOP 13 37
4 | ?
5 | ?
6 | ?
7 | ?
8 | ?
9 | STOP 13 37
```

Hierbei markiert das Fragezeichen (`?`) Zellen im Speicher, die vom Programm nicht gesetzt worden sind, also für die Betrachtung nicht relevant sind. Der Pfeil symbolisiert den als nächsten auszuführenden KI-Befehl.

KI-Befehl	**Arg A**	**Arg B**	**Beschreibung**
STOP			Stoppt die KI. Die KI hat verloren.
MOV_R	Source_R	Target_R	Kopiert den Inhalt der Speicherzelle *Source* in die Speicherzelle *Target*. Die Positionen sind relativ zum aktuellen KI-Befehl
MOV_I	Source_R	Target_I	Kopiert den Inhalt der Speicherzelle *Source* in die Speicherzelle *Target*. Die Position von *Source* ist relativ zum aktuellen KI-Befehl. Die Position von *Target* ist indirekt gegeben. D.h. es wird zunächst die Speicherzelle relativ zum aktuellen KI-Befehl (Intermidiate) bestimmt. Anschließend wird durch Eintrag B erneut relativ eine Speicherzelle bestimmt. Diese ist die Speicherzelle *Target*.
ADD	ValA	ValB	Addiert die Werte von *ValA* und *ValB* und speichert das Ergebnis in Eintrag B.
ADD_R	ValA	Target_R	Addiert den Wert von *ValA* mit dem Wert von Eintrag B der Speicherzelle *Target* und speichert das Ergebnis in Eintrag B *Target*. Die Position von *Target* ist relativ.
JMP	Target_R		Springt zu der Speicherzelle, die durch den Wert von *Target* gegeben ist. Die Position ist relativ zum aktuellen KI-Befehl.
JMZ	Target_R	$\text{CheckCell}_\text{R}$	Springt zu der Speicherzelle, die durch den Wert von *Target* gegeben ist, wenn der Wert von Eintrag B der Speicherzelle *CheckCell* 0 ist. Die Position von *Target* und *CheckCell* ist relativ zum aktuellen KI-Befehl.
CMP	First_R	Second_R	Vergleicht den Eintrag A der Speicherzelle *First* mit dem Eintrag B der Speicherzelle *Second*. Wenn die Werte ungleich sind, wird der nächste KI-Befehl übersprungen. Die Positionen sind relativ zum aktuellen KI-Befehl.
SWAP	First_R	Second_R	Vertauscht den Eintrag A von *First* mit dem Eintrag B von *Second*. Die Positionen *First* und *Second* sind relativ zum aktuellen KI-Befehl.

Tabelle A.1: Beschreibung der KI-Befehle

A.3 Spielphasen

Das Spiel besteht aus mehreren Phasen. Diese werden im Folgenden erklärt.

A.3.1 Initialisierungsphase

In der Initialisierungsphase wird zuerst der Speicher initialisiert und anschließend die KIs in den Speicher geladen. Für die Initialisierung des Speichers gibt es zwei Möglichkeiten:

- **INIT_MODE_RANDOM:** Der Speicher wird mit zufälligen Werten initialisiert.
- **INIT_MODE_STOP:** Der Speicher enthält nur *STOP*-KI-Befehle. Alle Einträge sind 0.

KIs sind Programme, die im Speicher ausgeführt werden. Die KIs werden in den Speicher geladen, indem die KI-Befehle mitsamt Argumenten der KIs in den Speicher kopiert werden. Die Position der KIs im Speicher ist hierbei durch die Größe des Speichers und die Anzahl der KIs gegeben. Die KIs werden gleichmäßig im Speicher verteilt. Es gilt also, dass der erste KI-Befehl der ersten KI (KI 0) in der Speicherzelle 0 ist. Der erste KI-Befehl der zweiten KI (KI 1) ist in der Speicherzelle $\lfloor \frac{Anzahl\ Speicherzellen}{Anzahl\ KIs} \rfloor$. Der erste KI-Befehl der dritten KI (KI 2) ist in der Speicherzelle $\lfloor 2 * \frac{Anzahl\ Speicherzellen}{Anzahl\ KIs} \rfloor$. Und so weiter. Wichtig ist, dass die KI-Befehle der KIs nicht überlappen. Die Ausführung der KIs beginnt immer mit dem ersten KI-Befehl der KI, der nicht *STOP* ist. Jede KI muss mindestens einen KI-Befehl haben, der nicht *STOP* ist.

A.3.2 Spielphase

In der Spielphase werden die KIs ausgeführt. Hierbei wird pro Zug immer ein KI-Befehl einer KI ausgeführt. Eine Runde ist beendet, wenn alle KIs, die einen KI-Befehl ausführen können, einen KI-Befehl ausgeführt haben. Die Reihenfolge der KIs ist hierbei fest und durch die initiale Position der KIs im Speicher gegeben. Die Reihenfolge ist also KI 0, KI 1, KI 2, usw. Wichtig ist, dass die KIs nicht parallel ausgeführt werden. Sollte eine KI den KI-Befehl *STOP* ausführen, so wird die KI gestoppt und die Ausführung der KI wird nicht fortgesetzt. Die KI ist somit aus dem Spiel und wird in den folgenden Runden nicht mehr ausgeführt. Sollte nur noch eine KI übrig sein, so dominiert diese KI zu diesem Zeitpunkt das Spiel. Beachten Sie, dass die KI weiterhin KI-Befehle ausführen kann, auch wenn sie bereits gewonnen hat, da die KI dominiert.

A.4 Implementierung des Spiels

In diesem Abschnitt wird die Implementierung des Spiels beschrieben. Das Spiel soll mit einer textuellen Schnittstelle realisiert werden.

Information

Hinweise zur Implementierung dieser Abschlussaufgabe
Beachten Sie bitte nochmals die Einschränkung der zu verwendenden Pakete am Beginn der Aufgabenstellung. Vermeiden Sie für diese Abschlussaufgabe insbesondere die Verwendung von Streams.

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.

A.4.1 Darstellung der Beispielinteraktionen

In Beispielinteraktionen stellt das Symbol `%>` (Prozent-Zeichen und Größer-Zeichen gefolgt von einem Leerzeichen) die Kommandozeile dar. Der Programmname ist frei gewählt und muss bei Ihnen nicht *CodeFight* lauten. Das Symbol `>` (Größer-Zeichen gefolgt von einem Leerzeichen) stellt eine Benutzereingabe dar und ist selbst nicht Teil der Eingabe. Sollte in einer Interaktion `[...]` geschrieben werden, so bedeutet dies, dass weitere Interaktionen ausgelassen worden sind, um auf eine spezielle Stelle der Interaktion einzugehen. `[...]` ist keine Ausgabe Ihres Spiels. Darstellungen in eckigen Klammern `[]`, wie `[KI Name]` stellen ebenso Platzhalter dar. Hierbei wird innerhalb der Klammern eine kurze Erklärung zum Verständnis eingefügt. Beachten Sie, dass solche Platzhalter niemals Teil der Ausgabe Ihres Programms sind. Zuletzt gibt es noch den Platzhalter ␣, welcher benutzt wird, um in ausgewählten Beispielinteraktionen ein Leerzeichen darzustellen. Dieses ist auch nie Teil der Ausgabe des Programms. Beachten Sie bitte, dass Sie die Interaktionen nicht aus diesem PDF kopieren sollten, da es beim Kopieren aus PDFs zu Veränderungen kommen kann. Verwenden Sie hierfür stattdessen die Textdateien im Ilias und fragen Sie bei Unklarheiten zum Format in den entsprechenden Foren nach.

A.4.2 Hinweis zum Begriff Befehl

Im folgenden Teil bezieht sich der Begriff *Befehl* auf unterschiedliche Aspekte des Programms. Unterscheiden Sie deshalb nochmals folgende Konzepte:

- **Speicherzelle:** (*auch: Befehl im Speicher*) ist ein Abschnitt im Speicher des Spiels mit Index.
- **KI-Befehl:** beschreibt im Allgemeinen den ausführbaren Inhalt einer Speicherzelle im Spiel. Wird zusätzlich noch auf die Einträge des KI-Befehls explizit eingegangen, so ist damit lediglich der Typ des KI-Befehls gemeint.
- **(KI-)Befehl der KI:** wird hauptsächlich im Kontext der Darstellung des Speichers verwendet, um durch die Zugehörigkeit zu vermitteln, von welcher KI ein **KI-Befehl** zuletzt bearbeitet wurde, oder als nächstes ausgeführt wird.
- **Interaktionsbefehl:** ist ein, in den folgenden Abschnitten beschriebener, Befehl, der vom menschlichen Benutzer Ihres Programms, zur Steuerung dessen, eingegeben wird.

A.4.3 Quit-Befehl

Generell muss Ihr Programm jederzeit mit der Eingabe des Befehls `quit` beendet werden können. Beachten Sie, dass für das Testen Ihrer Abgabe dieser Befehl essenziell ist, da viele der Tests `quit` am Ende einer Testsequenz verwenden, um Ihr Programm zu beenden. Stellen Sie daher sicher, dass der Befehl in jeder Situation funktioniert. Beachten Sie bitte nochmals, dass Sie hierfür nicht `System.exit()` oder andere ausgeschlossene Funktionen verwenden dürfen.

A.4.4 Programmstart

Ihr Programm wird mit mehreren Argumenten gestartet. Sollte die Anzahl der Argumente oder das Argument selbst nicht korrekt sein, so soll eine Fehlermeldung ausgegeben werden und das Programm beendet werden. Das erste Argument gibt die Größe des Speichers an. Dieser liegt im Bereich von $[7, 1337]$.

Die weiteren Argumente definieren Symbole, die in der Darstellung des Speichers verwendet werden. Die Symbole können jeweils aus mehreren Zeichen bestehen. Einzige Einschränkung ist hierbei, dass die Symbole keine Leerzeichen enthalten, oder mehrfach auftreten dürfen. Es werden zunächst 4 KI-unabhängige Symbole und anschließend 2 Symbole je KI definiert. Die Anzahl der KIs, die in der Simulation maximal gegeneinander antreten können, wird durch die Anzahl der Argumente bestimmt. Gleichzeitig müssen Symbole für mindestens 2 KIs definiert werden. Die vier KI-unabhängigen Symbole sind, in der Reihenfolge, in der sie definiert werden, die Symbole für:

[1] Einen seit Start unbearbeiteten KI-Befehl

[2] Die Bereichsgrenzen der Bereichsanzeige

[3] Den nächsten KI-Befehl der nächsten KI

[4] Die nächsten KI-Befehle aller anderen KIs

Die zwei Symbole je KI sind, in der Reihenfolge, in der sie definiert werden, die Symbole für:

a) Standard-Symbol der KI-Befehle der KI

b) KI-Bomben (spezielle KI-Befehle; siehe unten)

Die Symbole werden im entsprechenden Befehl A.4.7.2 verwendet.

Sollte der Programmstart erfolgreich sein, bestätigt dies eine Begrüßung, wie sie in der folgenden Beispielinteraktion beschrieben ist.

Beispielinteraktion

```
%> java CodeFight 1337 # ? _ ^ G g B b
Welcome to CodeFight 2024. Enter 'help' for more details.
[...]
```

A.4.5 Befehl: help

Der Befehl `help` gibt zeilenweise eine kurze Beschreibung der Befehle aus, die in der aktuellen Phase des Spiels verfügbar sind. Die Beschreibung der Befehle ist von Ihnen zu wählen, sollte aber, wie alle Ausgaben des Programms, auf Englisch sein. Die Beschreibung soll ausreichend sein, um die Befehle zu verstehen. Sie darf keine Zeilenumbrüche enthalten. Die Reihenfolge der Befehle ist hierbei aufsteigend (lexikografisch nach Name des Befehls). Das Format ist immer `[Name des Befehls]: [Beschreibung]`.

Beispielinteraktion

```
[...]
> help
[...]
add-ai: Add some valuable description here.
help: Add some valuable description here.
[...]
```

A.4.6 Initialisierungsphase

In der Initialisierungsphase können verschiedene Befehle ausgeführt werden. Diese dienen dazu, die KIs zu laden und den Speicher zu initialisieren. Die für diese Phase relevanten Befehle werden im Folgenden beschrieben.

A.4.6.1 Befehl: add-ai Der Befehl `add-ai` registriert eine KI mit dem angegebenen Namen und den angegebenen KI-Befehlen. Registrierte KIs können anschließend gegeneinander spielen. Die KI-Befehle und auch die Argumente sind durch Kommata getrennt. Generell ist die Syntax des Befehls wie folgt: `add-ai [Name der KI] [Liste der KI-Befehle]`. Es ist nicht erlaubt, eine bestehende KI (definiert durch den Namen) zu überschreiben. Bei der Namenswahl der KI ist zu beachten, dass diese im Namen keine Leerzeichen enthalten darf. Das erfolgreiche Registrieren einer KI wird durch die Ausgabe des Namens der KI bestätigt.

Beispielinteraktion

```
[...]
> add-ai Sleepy ADD,10,-1,MOV_I,2,-1,JMP,-2,0,STOP,13,37
Sleepy
[...]
```

A.4.6.2 Befehl: remove-ai Der Befehl `remove-ai` entfernt eine KI mit dem angegebenen Namen. Es ist nicht erlaubt, eine nicht existierende KI zu entfernen. Die Syntax des Befehls ist wie folgt: `remove-ai [Name der KI]`. Das erfolgreiche Entfernen einer KI wird durch die Ausgabe des Namens der KI bestätigt.

Beispielinteraktion

```
[...]
> remove-ai Sleepy
Sleepy
[...]
```

A.4.6.3 Befehl: set-init-mode Der Befehl `set-init-mode` setzt den Modus, mit dem der Speicher initialisiert wird. Wie oben beschrieben, gibt es zwei Modi: `INIT_MODE_RANDOM` und `INIT_MODE_STOP`. Die Syntax des Befehls ist wie folgt: `set-init-mode [Modus] [Optionale Parameter]`. Falls der Befehl nicht benutzt worden ist, soll der Modus `INIT_MODE_STOP` sein. Für diesen Modus sind keine Parameter notwendig. Für den Modus `INIT_MODE_RANDOM` ist ein Parameter nötig. Dieser ist eine Granzzahl zwischen $[-1337, 1337]$ und beschreibt den *Seed*, der für die Initialisierung des Speichers verwendet wird. Die Beschreibung, wie genau der Speicher in diesem Fall initialisiert wird, finden Sie in A.4.6.3. Die Ausgabe des Befehls enthält die Änderung des Modus, sofern der Modus (oder dessen Parameter) sich geändert haben. Das Format der Ausgabe ist hierbei: `Changed init mode from [Alter Modus] to [Neuer Modus]`. Sollte sich der Modus nicht geändert haben, so wird nichts ausgegeben.

Beispielinteraktion

```
[...]
> set-init-mode INIT_MODE_RANDOM 0
Changed init mode from INIT_MODE_STOP to INIT_MODE_RANDOM 0
> set-init-mode INIT_MODE_STOP
Changed init mode from INIT_MODE_RANDOM 0 to INIT_MODE_STOP
[...]
```

INIT_MODE_RANDOM mit Seed Für die Initialisierung des Speichers mit dem Modus `INIT_MODE_RANDOM` ist der Parameter `Seed` notwendig. Dieser wird zur Erzeugung eines Zufallszahlengenerators verwendet.

Gehen Sie bei der Initialisierung des Speichers wie folgt vor:

1. Verwenden Sie den Konstruktor `Random(long seed)`[6], um einen neuen Zufallszahlengenerator mithilfe des *Seeds* zu instanziieren.
2. Für jede Speicherzelle beginnend bei der Speicherzelle 0 gehen Sie nun wie folgt vor:
   a. Verwenden Sie die Funktion `Random::nextInt(int bound)` um eine Zufallszahl im Bereich von $[0, \textit{Anzahl der KI} - \textit{Befehle})$ zu erzeugen. Und setzen Sie den KI-Befehl der Speicherzelle auf den KI-Befehl, der an der Stelle der Zufallszahl in der Liste der KI-Befehle (A.1) steht.
   b. Verwenden Sie die Funktion `Random::nextInt()` um eine Zufallszahl für Eintrag A zu erzeugen und setzen Sie den Eintrag A der Speicherzelle auf diese Zufallszahl.
   c. Verwenden Sie die Funktion `Random::nextInt()` um eine Zufallszahl für Eintrag B zu erzeugen und setzen Sie den Eintrag B der Speicherzelle auf diese Zufallszahl.

A.4.6.4 Befehl: start-game Der Befehl `start-game` startet das Spiel. Hierbei ist das Format des Befehls wie folgt: `start-game [Liste der KIs]`. Die Liste der KIs ist eine durch Leerzeichen getrennte Liste von KI-Namen. Das Spiel kann nur dann gestartet werden, wenn mindestens zwei KIs gewählt sind. Es ist erlaubt mehrfach die gleiche KI zu wählen. Falls eine KI mehrfach instanziiert wird, so wird der Name der jeweiligen KI als `[Name der KI]#[Nummer der KI beginnend bei 0]` gesetzt. Die Reihenfolge der KIs ergibt sich aus der Reihenfolge der KIs in der übergebenen Liste. Beachten Sie, dass die Anzahl der KIs, die maximal gegeneinander antreten können, durch die Anzahl der Argumente beim Programmstart bestimmt wird. Sollte das Spiel gestartet werden und damit die Initialisierungsphase verlassen werden, so quittiert das Programm den Befehl mit der Ausgabe `Game started.` Damit beginnt die Spielphase.

Beispielinteraktion

```
[...]
> start-game Sleepy HelloWorld
Game started.
[...]
```

A.4.7 Spielphase

In der Spielphase können verschiedene Befehle ausgeführt werden. Diese dienen dazu die KIs auszuführen und den aktuellen Zustand des Spiels anzuzeigen. Die für diese Phase relevanten Befehle werden im Folgenden beschrieben.

A.4.7.1 Befehl: next Der Befehl `next` führt die nächsten KI-Befehle der KIs aus. Hierbei ist das Format des Befehls wie folgt: `next [Anzahl der KI-Befehle]`. Die Anzahl der KI-Befehle ist optional und standardmäßig 1. Die Anzahl gibt hierbei an, wie viele KI-Befehle insgesamt ausgeführt werden sollen. Zu Beginn eines Spiels mit zwei KIs würde der Befehl `next 2` also zwei KI-Befehle ausführen: Einen KI-Befehl der ersten KI und einen KI-Befehl der zweiten KI. KIs, die

[6] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Random.html`

durch das Ausführen des KI-Befehls `STOP` gestoppt wurden, werden nicht mehr ausgeführt. Sollte also im Beispiel der zwei KIs eine KI durch den ersten KI-Befehl gestoppt werden, so würde bei der Ausführung des Befehls `next 2` als nächstes die verbleibende KI zwei KI-Befehle ausführen. Sollten alle KIs gestoppt sein, so kann weiterhin der Befehl `next` ausgeführt werden. In diesem Fall werden dann keine KI-Befehle tatsächlich mehr ausgeführt.

Sollte einer der KI-Befehle, der durch den `next`-Befehl ausgeführt wird, ein *STOP*-KI-Befehl sein, so wird die KI gestoppt und die Ausführung der KI wird (wie schon oben beschrieben) nicht fortgesetzt. Ein solches Ereignis wird durch die Ausgabe `[Name der KI] executed [Zähler] steps until stopping.` quittiert. Der Zähler gibt an, wie viele KI-Befehle die KI seit Spielbeginn ausführen konnte, bis sie gestoppt wurde. Der STOP-Befehl zählt hierbei nicht mit. Alle anderen ausgeführten KI-Befehle erzeugen keine Ausgabe. Nach dem Quittieren des KI-Befehls, läuft die Ausführung des `next`-Befehls normal weiter. D.h., die (potentiell) verbleibenden Schritte werden durch die verbleibenden KIs ausgeführt.

Beispielinteraktion

```
[...]
> next 2
> next 2000
SleepyII executed 32 steps until stopping.
MainAI executed 42 steps until stopping.
[...]
```

A.4.7.2 Befehl: show-memory Der Befehl `show-memory` zeigt den aktuellen Zustand des Speichers an. Es wird zwischen der *Speicheranzeige* und der *Bereichsanzeige* unterschieden.

In der Darstellung des Speichers werden die Symbole verwendet, die Sie zum Programmstart als Kommandozeilenargumente übergeben bekommen haben.

Um eine Speicherzelle zu repräsentieren, wird das Symbol mit der höchsten Priorität (weiter oben in der folgenden Liste) gewählt, das für den KI-Befehl dieser Zelle zutreffend ist. Die Prioritäten sind hierbei wie folgt:

1. Ist nächster KI-Befehl der nächsten KI
2. Ist nächster KI-Befehl aller anderen KIs
3. Ist eine KI-Bombe
4. Ist ein KI-Befehl einer KI
5. Ist seit Start unbearbeitet

Befehle einer KI sind alle KI-Befehle, die von dieser KI beim Initialisieren in den Speicher übernommen wurden (unabhängig davon, was, vor der Initialisierung der KI, in der Speicherzelle stand). Im weiteren Verlauf des Spiels ändert sich die Zugehörigkeit von KI-Befehlen zu der KI, die diesen KI-Befehl zuletzt in irgendeiner Form bearbeitete. Die Einordnung als KI-Bombe, oder nicht, wird hierbei gegebenenfalls ebenso aktualisiert. Für eine Bearbeitung müssen die Einträge des KI-Befehls nicht verändert worden sein. Es reicht, dass eine KI die Zelle verändern/beschreiben wollte.

Eine KI-Bombe ist ein gewöhnlicher KI-Befehl einer KI, der eine der folgenden Bedingungen erfüllt:

a) Der KI-Befehl ist ein `STOP`-KI-Befehl.

b) Der KI-Befehl ist ein `JMP`-KI-Befehl, mit Eintrag A = 0.

c) Der KI-Befehl ist ein `JMZ`-KI-Befehl, mit Eintrag A = 0 und Eintrag B = 0.

Das Format des Befehls für die Speicheranzeige ist wie folgt: `show-memory`. Diese zeigt einen Überblick des gesamten Speichers an. Beginnend mit der ersten Speicherzelle, gefolgt von der Nächsten, usw., werden die repräsentierenden Symbole direkt konkateniert und als eine Zeile ausgegeben.

Im folgenden Ausschnitt ist somit zu erkennen, dass der 3. KI-Befehl im Speicher, der ist, der als nächstes ausgeführt werden wird. Danach muss der KI-Befehl an der 19. Stelle im Speicher folgen (nur eindeutig ersichtlich, da durch gegebene Kommandozeilenargumente bekannt ist, dass nicht mehr als zwei KIs gleichzeitig verwendet werden können). Die KI-Befehle an Position 1, 2, 4 und 8 im Speicher, wurden zuletzt von der ersten KI bearbeitet (oder initialisiert), wobei es sich lediglich an Position 8 um eine Bombe handelt. Gleiches gilt an den Positionen 17, 18, 20 und 27 für die zweite KI, mit KI-Befehl 27 als Bombe. Alle anderen KI-Befehle des Speichers wurden seit Spielbeginn von keiner KI bearbeitet.

Hinweis: Durch die Formulierung wird ersichtlich, dass mit diesem Befehl allein keine Annahmen über die KIs direkt getroffen werden können. Es werden lediglich die Befehle im Speicher selbst betrachtet. Deren Informationen über Bearbeitung und geplanter Ausführung geben dennoch einen guten Überblick über das Verhalten der KIs.

Beispielinteraktion

```
%> java CodeFight 32 # ? _ ^ G g B b
[...]
> show-memory
GG_G###g########BB^B######b#####
[...]
```

Das Format des Befehls für die Bereichsanzeige ist wie folgt: `show-memory [Speicherzelle]`. Die Bereichsanzeige zeigt für einen Bereich von 10 Speicherzellen, beginnend bei der, durch die Nummer, angegebenen Speicherzelle, den Zustand an. Dazu wird zunächst die Speicheranzeige ausgegeben, mit der Änderung, dass das Symbol der Bereichsgrenzen so, je **zwischen** zwei Befehlssymbolen, eingefügt wird, so dass ein Bereich aus Befehlssymbolen zwischen diesen Grenzen entsteht, der genau dem Speicherbereich entspricht, der angezeigt werden soll. Die geänderte Darstellung der Speicheranzeige ist somit um `2 * (Länge des Bereichsgrenzensymbols)` länger, als sie es normalerweise wäre. Darauf folgen, zeilenweise, die einzelnen Speicherzellen des anzuzeigenden Bereichs. Jede Darstellung einer Speicherzelle entspricht dem Format `[Symbol] [Index]: [KI-Befehl] | [Eintrag A] | [Eintrag B]`. Alle Angaben der einzelnen Platzhalter sollen rechtsbündig, entsprechend ihrer Spalte, ausgegeben werden. Beachten Sie auch die Sonderzeichen (inklusive Leerzeichen), Platzhalter ausgenommen, im Format.

Beispielinteraktion

```
%> java CodeFight 32 # ? _ ^ G g B b
[...]
> show-memory 17
GG_G###g########B?B^B######b?#####
B 17: MOV_I |  2 | -1
^ 18:   JMP | -2 |  0
B 19:  STOP | 13 | 37
# 20:  STOP |  0 |  0
# 21:  STOP |  0 |  0
# 22:  STOP |  0 |  0
# 23:  STOP |  0 |  0
# 24:  STOP |  0 |  0
# 25:  STOP |  0 |  0
b 26:  STOP | 13 | 37
[...]
```

A.4.7.3 Befehl: show-ai Der Befehl `show-ai` zeigt den aktuellen Zustand einer KI an. Hierbei ist das Format des Befehls wie folgt: `show-ai [Name der KI]`. Die Ausgabe des Befehls besteht aus maximal zwei Zeilen:

```
[Name der KI] [RUNNING|STOPPED]@[Zähler der KI]
Next Command: [KI-Befehl, der als nächstes ausgeführt wird] @[Speicherzelle]
```

Die erste Zeile gibt den Namen der KI an, gefolgt von dem Status der KI und dem Zähler der KI. Der Zähler der KI gibt an, wie viele KI-Befehle die KI seit Spielbeginn ausgeführt hat (ggf. inklusive STOP). Die zweite Zeile gibt den nächsten KI-Befehl, sowie die Nummer der Speicherzelle, in der sich der KI-Befehl befindet, an. Sollte die KI gestoppt sein, so wird die zweite Zeile nicht ausgegeben.

Beispielinteraktion

```
[...]
> show-ai Sleepy
Sleepy (RUNNING@0)
Next Command: ADD_R|10|-2 @2
[...]
> show-ai Sleepy
Sleepy (STOPPED@42)
[...]
```

A.4.7.4 Befehl: end-game Der Befehl `end-game` beendet das Spiel. Hierbei ist das Format des Befehls wie folgt: `end-game`. Am Ende des Spiels werden die Listen der laufenden und gestoppten KIs angezeigt. Hierbei werden nur Listen mit mindestens einer KI ausgegeben. Die Ausgabe ist wie folgt: `Running AIs: [Namen laufender KIs durch Leerzeichen getrennt]` und `Stopped AIs: [Namen gestoppter KIs durch Leerzeichen getrennt]`. Die Listen sind hierbei aufsteigend

nach der initialen Position der KIs im Speicher sortiert. Nach dem Ausführen des Befehls befindet sich das System wieder in der Initialisierungsphase.

Beispielinteraktion

```
[...]
> end-game
Running AIs: Sleepy#0, MagicAI
Stopped AIs: HelloWorld, Sleepy#1
[...]
```

A.5 Beispielinteraktionen

Beispielinteraktion

```
%> java CodeFight 32 # ? _ ^ G g B b
Welcome to CodeFight 2024. Enter 'help' for more details.
> add-ai Dwarf ADD_R,4,3,MOV_I,2,2,JMP,-2,0,STOP,0,0
Dwarf
> add-ai Sleepy ADD,10,-1,MOV_I,2,-1,JMP,-2,0,STOP,13,37
Sleepy
> start-game Dwarf Sleepy
Game started.
> show-memory
_GGG############^BBB############
> next 50
Sleepy executed 12 steps until stopping.
Dwarf executed 24 steps until stopping.
> show-memory 16
gGGg###g###ggb##?gBBBg###gb?##gb##
g 16:  STOP | 13 | 45
B 17: MOV_I |  2 | -1
B 18:   JMP | -2 |  0
B 19:  STOP | 13 | 37
g 20:  STOP | 13 | 49
# 21:  STOP |  0 |  0
# 22:  STOP |  0 |  0
# 23:  STOP |  0 |  0
g 24:  STOP | 13 | 53
b 25:  STOP | 13 | 37
> end-game
Stopped AIs: Dwarf, Sleepy
> add-ai Caterpillar MOV_R,0,1
Caterpillar
> add-ai Pogo JMP,0,0
Pogo
> set-init-mode INIT_MODE_RANDOM 28
Changed init mode from INIT_MODE_STOP to INIT_MODE_RANDOM 28
```

Beispielinteraktion

```
> start-game Pogo Caterpillar
Game started.
> next 181
> show-memory 9
BBBBBBBBB?B_^BBBBBBB?BBBBBBBBBBBBB
B  9: MOV_R | 0 | 1
_ 10: MOV_R | 0 | 1
^ 11: MOV_R | 0 | 1
B 12: MOV_R | 0 | 1
B 13: MOV_R | 0 | 1
B 14: MOV_R | 0 | 1
B 15: MOV_R | 0 | 1
B 16: MOV_R | 0 | 1
B 17: MOV_R | 0 | 1
B 18: MOV_R | 0 | 1
> show-ai Caterpillar
Caterpillar (RUNNING@90)
Next Command: MOV_R|0|1 @10
> show-ai Pogo
Pogo (RUNNING@91)
Next Command: MOV_R|0|1 @11
> end-game
Running AIs: Pogo, Caterpillar
> quit
```