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

Übungsblatt 4 Version 1.0	20 Punkte	Ausgabe:	11.12.2024, ca. 12:00 Uhr
		Abgabestart:	18.12.2024, 12:00 Uhr
		Abgabefrist:	09.01.2025, 06:00 Uhr

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.

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.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

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 Übungsblatt 4 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` und `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 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.

[4] `https://artemis.praktomat.cs.kit.edu/`

- 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 18.12.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 09.01.2025, 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.
- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe B 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.

Aufgabe A: Perseverance (15 Punkte)

Mit Personen besetzte Mars-Missionen erscheinen heute noch wie Science-Fiction, jedoch peilt die NASA erste Missionen im Jahr 2040 an, SpaceX sogar schon 2029, wie die Präsidentin des Unternehmens im Mai 2022 bestätigte. Die Frage nach dem Leben auf dem Mars ist allerdings kein modernes Phänomen, sondern beflügelt die Fantasie der Menschen schon seit Jahrhunderten:

Schon im 17. Jahrhundert wurden die Polkappen des Mars entdeckt. Im späten 18. Jahrhundert beschrieb William Herschel erstmals ein Auf- und Absinken der Polkappen mit Wechsel der Jahreszeiten auf der jeweiligen Hemisphäre. Außerdem beobachteten die Menschen dunkle Flecken auf der Marsoberfläche, die ihre Farbe änderten und wuchsen oder schrumpften. Man hielt sie für ausgedehnte Vegetationszonen, deren Ausdehnung sich mit den Jahreszeiten änderte.

Als im 19. Jahrhundert Astronomen erkannten, dass der Mars Ähnlichkeiten zur Erde aufweist (Länge eines Tages, Achsneigung) und Teleskopbeobachtungen sogenannte Marskanäle zeigten explodierten die Spekulationen über Leben auf dem Mars förmlich. Später stellte sich dann allerdings heraus, dass es sich bei den Marskanälen um eine optische Täuschung gehandelt hatte. Heutzutage geht die Wissenschaft davon aus, dass es kein höheres Leben auf dem Mars gibt. Jedoch werden primitive Lebensformen tiefer im Boden für möglich gehalten.

Um die Eigenschaften des Mars vor Ort zu erforschen, werden sogenannte Rover eingesetzt. Die erste erfolgreiche Landung gelang der NASA 1997 mit dem Sojourner der Pathfinder-Mission. Der im Februar 2021 gelandete Rover Perseverance beschäftigt sich explizit mit der Frage, ob früher oder zukünftig Leben auf dem Mars möglich wäre.

Unglücklicherweise kam es vor einigen Wochen zu einem Unfall, bei dem mehrere Systeme ausgefallen sind[5]. Inzwischen konnte wieder eine Verbindung zum Rover aufgenommen werden, jedoch haben die Analysen ergeben, dass das Navigationssystem weiterhin defekt ist.

Ihre Aufgabe besteht darin ein Programm zu entwickeln, das die Aufgabe des defekten Navigationssystems übernimmt. Der Rover kommuniziert dafür über die Standardein- und Ausgabe mit Ihrem Programm.

A.1 Kommunikationsprotokoll

Der Rover steuert Ihr Navigationssystem über eine Reihe von Befehlen. Die folgende Notation wird zur Beschreibung der Befehle verwendet.

- Spitze Klammern `<...>` stellen Parameter der Befehle dar.
- Eckige Klammern `[...]` schließen optionale Parameter ein und werden vom Rover nicht zwingend gesendet.
- Geschweifte Klammern `{Befehl1, Befehl2, ...}` beschreiben kommagetrennt eine Reihe von Befehlen mit gleichem Schema.

[5] Gerüchten nach war die verantwortliche Person beim Eingeben neuer Koordinaten etwas zu stark durch das Hören von „Talking To The Moon“ abgelenkt.

Sollte ein Befehl zu einem Zeitpunkt keinen Sinn ergeben, da zum Beispiel noch keine Aufnahme (siehe A.1.1) übertragen worden ist, so signalisieren Sie dies durch Ausgabe der Nachricht `ERROR`.

Wir empfehlen Ihnen, in dieser Aufgabe das **Command-Pattern** zu benutzen. Ein Beispiel dafür finden sie im Wiki [6].

A.1.1 `new <width> <height>`

Das visuelle System wurde durch Ausschalten und wieder Einschalten glücklicherweise wieder in Gang gebracht. Der Rover ist daher in der Lage ein Abbild der Umgebung aufzunehmen. Mit dem Befehl `new <width> <height>` wird die Übertragung einer solchen Aufnahme eingeleitet.

Auf den Befehl folgen dann in der Regel (siehe unten) `<height>` Zeilen der Länge `<width>`, die das Terrain beschreiben. Folgende Zeichen können in einer gültigen Aufnahme auftreten:

- `R`: Beschreibt die Rover-Position.
- `x`: (kleines x) Beschreibt das Ziel: Ein Ort, an dem sich möglicherweise ein Zeichen von Leben befinden könnte.
- `X`: (großes X) Ein spezielles Zeichen, falls sich Rover und Ziel auf demselben Feld befinden.
- `|, _, /, \, *`: Beschreiben Hindernisse, die vom Rover nicht passiert werden können.
- Leerzeichen (im Folgenden auch als ␣ dargestellt): Beschreibt vom Rover passierbares Terrain.

Eine erfolgreiche Übertragung könnte folgendermaßen aussehen (links mit sichtbaren Leerzeichen; rechts ohne, für bessere Lesbarkeit):

```
␣␣␣␣␣__________␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣/␣␣␣␣␣␣␣␣␣␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣\_______/␣␣␣␣␣␣␣␣␣␣␣___␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣*␣␣␣␣x␣␣␣␣␣␣␣␣␣**␣␣␣␣|␣␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣␣␣___␣␣␣␣␣**␣␣␣␣␣␣␣\␣␣␣␣␣/␣␣␣␣␣␣␣␣␣
␣␣␣␣␣/␣␣␣␣\␣␣␣␣␣**␣␣␣␣␣␣␣␣␣\__|␣␣␣␣␣R␣␣␣␣␣
␣␣␣␣|␣␣␣␣␣␣\_␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣\␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣\____|␣␣␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣*␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

```
  __________
 /          \
|           /
|          |
 \_______/
                 ___
                /   |
*    x        **   |    |
**                /     |
  ___      **     \    /
 /   \     **       \__|     R
|     \_
 \      /
  \____|   *
           * *
```

[6] `https://sdq.kastel.kit.edu/programmieren/Command_Pattern`

Bei einer so großen Übertragung von Mars zu Erde kommt es gelegentlich auch zu Fehlern. Stellen Sie sicher, dass Ihr Programm, sobald es eine Unstimmigkeit erkennt, dem Rover einmalig die Nachricht `ERROR` sendet. Aufgrund von den Latenzen wird der Rover nach der Fehlermeldung möglicherweise weitere Zeilen des Terrains übertragen, bevor er mit einem erneuten `new`-Befehl die Übertragung neu startet. Es werden allerdings nie mehr als `<height>` Zeilen übertragen.

Mögliche Fehler beinhalten zum Beispiel das Empfangen von unspezifizierte Zeichen, das Fehlen von Zeichen, inklusive dem Fehlen der Rover-Position oder dem Ziel und das Fehlen von ganzen Zeilen (d.h. ein anderer Befehl wird angefragt, obwohl nicht alle erwarteten Zeilen angekommen sind).

Wenn das empfangene Terrain plausibel ist, können Sie davon ausgehen, dass alles korrekt übertragen wurde.

A.1.2 `{up, down, left, right} [<count>]`

Um Bandbreite zu sparen, sendet der Rover vor dem Ausführen einer Bewegung die geplante Bewegunsrichtung, anstatt nach der Bewegung ein neues Terrain mittels dem `new`-Befehl zu übertragen.

Der Parameter `<count>` gibt die Anzahl der Felder an, die der Rover sich in Richtung {oben, unten, links, rechts} bewegen will. Der Standardwert des Parameters ist eins.

Die Übertragung ist bei dieser kurzen Nachricht immer fehlerfrei. Jedoch ist die Kollisionserkennung weiterhin gestört und daher kann es sein, dass der Rover bei der Ausführung der Bewegung mit einem Hindernis kollidiert. Berücksichtigen Sie dies in Ihrem Programm, indem Sie die Bewegung nur bis zur Kollision mit einem Hindernis verfolgen. Der Rover erkennt, wenn er das erfasste Gebiet verlassen würde, und stoppt auf dem letzten Feld, das noch zum Gebiet gehört.

A.1.3 `debug`

Der `debug`-Befehl wird nicht vom Rover gesendet, sondern ist für den Einsatz durch die Entwickler des Navigationssystems gedacht. Nach Eingabe des Befehls soll das Terrain inklusive Ziel und aktueller Rover-Position im Format des `new`-Befehls ausgegeben werden. Achten Sie darauf auch nachfolgende Leerzeichen am Ende einer Zeile mit auszugeben.

A.1.4 `path`

Der `path`-Befehl ist das Herzstück des Navigationssystems. Nach Aufruf des Befehls soll ein Weg von der aktuellen Rover-Position `R` zum Ziel `x` berechnet werden. In A.2 wird eine Möglichkeit zur Berechnung eines solchen Weges beschrieben, Sie dürfen aber auch ein anderes Verfahren implementieren. In den meisten Fällen wird es mehrere Wege zum Ziel geben, dann genügt die Berechnung eines beliebigen Weges. Allerdings muss für das gleiche Terrain und die gleiche Rover-Position immer der gleiche Weg ausgegeben werden.

Gibt es einen Weg zum Ziel, wird zuerst eine Zeile `PATH <n>` gesendet, gefolgt von n Zeilen der Form `{up, down, left, right} [<count>]`, wie in A.1.2 beschrieben. Sie dürfen, müssen aber

nicht, Befehle durch Verwendung des `<count>` Parameters zusammenfassen. Nach Ausführung der gesendeten Befehle muss der Rover ohne Kollision mit einem Hindernis am Ziel angelangt sein. Für den besonderen Fall, dass sich Rover und Ziel auf dem gleichen Feld befinden muss `PATH 0` ausgegeben werden.

Gibt keinen Weg zum Ziel, wird die Nachricht `FAIL` gesendet.

A.1.5 `debug-path`

Wie schon der `debug`-Befehl ist auch der `debug-path`-Befehl für den Einsatz durch die Entwickler gedacht. Der Befehl verhält sich wie der `debug`-Befehl, es wird jedoch zusätzlich der vom `path`-Befehl generierte Pfad visualisiert. Hierfür werden alle Felder, die der Rover durch das Ausführen der von `path` berechneten Befehle betreten würde, mit einem Punkt `.` markiert. Die Positionen des Rovers und des Ziels werden unverändert dargestellt.

Ist die `path`-Berechnung nicht erfolgreich, dann wird statt des Terrains die Nachricht `FAIL` ausgegeben.

Die Ausgabe könnte für das Terrain oben z.B. folgendermaßen aussehen:

```
␣␣␣␣␣__________␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣/␣␣␣␣␣␣␣␣␣␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣\_______/␣␣␣␣␣␣␣␣␣␣␣___␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣*␣␣␣␣x....␣␣␣␣**␣␣␣|␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣**␣␣␣␣␣␣␣.......␣␣/␣␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣␣␣___␣␣␣␣␣**␣␣␣.␣␣\␣␣␣␣/␣␣␣␣␣␣␣␣␣
␣␣␣␣/␣␣␣\␣␣␣␣**␣␣␣....\__|␣␣␣␣␣R.␣␣␣
␣␣␣|␣␣␣␣␣\_␣␣␣␣␣␣␣␣␣␣............␣␣␣
␣␣␣␣\␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣\____|␣␣␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣*␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
```

```
     __________
    /          \
   |          /
   |         |
    \_______/           ___
                       /   |
   *    x....    **   |    |
   **       .......  /     |
     ___     **   .  \    /
    /   \    **   ....\__|     R.
   |     \_          ............
    \      /
     \____|   *
             * *
```

A.1.6 `quit`

Ihr Programm muss sich zu jedem Zeitpunkt durch den `quit`-Befehl beenden lassen.

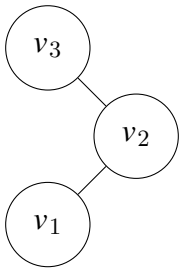
A.2 Berechnung eines Weges zum Ziel

Die Berechnung eines (kürzesten) Weges ist ein klassisches Problem der *Graphentheorie*.

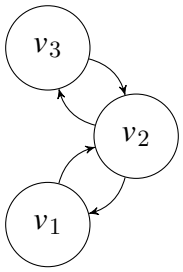
A.2.1 Graphentheorie

Ein *Graph* ist eine Struktur, die eine Menge von Objekten und die Verbindungen zwischen diesen Objekten repräsentiert. Die Objekte werden als *Knoten* $V = \{v_1, \ldots, v_n\}$ (engl. Vertices), die Verbindungen als *Kanten* $E = \{e_1, \ldots, e_m\}$ (engl. Edges) bezeichnet. Kanten können entweder gerichtet $e_i = (v_p, v_q)$ oder (wie in unserem Fall) ungerichtet $e_i = \{v_p, v_q\}$ sein, wobei $v_p, v_q \in V$. Ein Graph ist also ein Tupel aus der Knoten- und Kantenmenge $G = (V, E)$.

Beispiele für Graphen sind z.B. U-Straßenbahnnetze, Straßenkarten oder auch Stammbäume.



(a) Ein ungerichteter Graph $G = (V, E)$ mit $V = \{v_1, v_2, v_3\}$ und $E = \left\{\{v_1, v_2\}, \{v_2, v_3\}\right\}$.



(b) Ein gerichteter Graph $G = (V, E)$ mit $V = \{v_1, v_2, v_3\}$ und $E = \left\{(v_1, v_2), (v_2, v_3), (v_2, v_1), (v_3, v_2)\right\}$.

Viele Algorithmen benötigen auch bei ungerichteten Graphen eine gerichtete Darstellung. In diesem Fall werden ungerichtete Kanten durch zwei gerichtete Kanten (wie in A.1b) dargestellt.

A.2.2 Algorithmus

Im ersten Schritt wird das Terrain als Graph dargestellt. Hierbei können die Felder als Knoten modelliert werden. Die Kanten verbinden benachbarte Felder, falls der Rover beide Felder befahren darf.

Im zweiten Schritt wird eine *Breitensuche* ausgeführt. Informieren Sie sich selbständig über den Algorithmus und implementieren Sie diesen in ihr Navigationssystem.

A.3 Beispielinteraktion

Hinweis: Es gibt es viele verschiedene gültige Interaktionen, die sich durch die Implementierung des `path`-Befehls unterscheiden. Solange Ihre Abgabe einen gültigen Pfad generiert, wird die Abgabe von Artemis akzeptiert. Leerzeichen werden im Folgenden als ␣ dargestellt, die tatsächliche Ein- und Ausgabe in Artemis verwendet reguläre (unsichtbare) Leerzeichen. Eine Version der Beispielinteraktion ohne explizite Darstellung der Leerzeichen finden Sie im Ilias als Textdatei.

Beispielinteraktion

```
>new␣36␣15
>␣␣␣␣␣__________␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣/␣␣␣␣␣␣␣␣␣␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣|␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣|␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣\_______/␣␣␣␣␣␣␣␣␣␣␣___␣␣␣␣␣␣␣␣␣
>␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣|␣␣␣␣␣␣␣␣
>␣␣␣*␣␣␣␣x␣␣␣␣␣␣␣␣**␣␣␣␣|␣␣␣␣␣|␣␣␣␣␣␣␣␣
>␣␣␣**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣|␣␣␣␣␣␣␣␣
>␣␣␣␣␣␣___␣␣␣␣␣␣**␣␣␣␣␣␣␣\␣␣␣␣␣/␣␣␣␣␣␣␣␣
>␣␣␣␣␣/␣␣␣␣\␣␣␣␣**␣␣␣␣␣␣␣␣\__|␣␣␣␣␣␣R␣␣␣␣␣
>␣␣␣|␣␣␣␣␣␣\_␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣␣\␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣␣␣\____|␣␣␣␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>up␣4
>down
>left␣10
>debug
␣␣␣␣␣__________␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣/␣␣␣␣␣␣␣␣␣␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣\_______/␣␣␣␣␣␣␣␣␣␣␣___␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣*␣␣␣␣x␣␣␣␣␣␣␣␣**␣␣␣␣|␣␣␣␣␣|R␣␣␣␣␣␣␣
␣␣␣**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣␣␣␣___␣␣␣␣␣␣**␣␣␣␣␣␣␣\␣␣␣␣␣/␣␣␣␣␣␣␣␣
␣␣␣␣␣/␣␣␣␣\␣␣␣␣**␣␣␣␣␣␣␣␣\__|␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣\_␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣\␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣\____|␣␣␣␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣*␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>path
PATH␣9
up␣2
left
up
left␣5
down
left␣2
down
left␣12
down
```

Beispielinteraktion

```
>debug-path
␣␣␣␣␣__________␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣/␣␣␣␣␣␣␣␣␣␣\␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣|␣␣␣␣␣␣␣␣␣␣|␣␣␣␣␣␣␣␣␣.....␣␣␣␣␣␣␣␣
␣␣␣␣\_______/␣␣␣␣␣␣␣␣...␣___..␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣.............␣␣/␣␣␣|.␣␣␣␣␣␣␣␣
␣␣␣*␣␣␣␣x␣␣␣␣␣␣␣␣**␣␣␣|␣␣␣␣|R␣␣␣␣␣␣␣␣
␣␣␣**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣/␣␣␣␣␣|␣␣␣␣␣␣␣␣
␣␣␣␣␣␣___␣␣␣␣␣␣**␣␣␣␣␣␣␣\␣␣␣␣/␣␣␣␣␣␣␣␣
␣␣␣␣␣/␣␣␣␣\␣␣␣␣␣**␣␣␣␣␣␣␣␣\__|␣␣␣␣␣␣␣␣␣
␣␣␣␣|␣␣␣␣␣␣\_␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣\␣␣␣␣␣␣/␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣\____|␣␣␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣*␣*␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
>quit
```

Aufgabe B: Dots-and-Boxes Erweiterung (5 Punkte)

In dieser Aufgabe sollen das Spiel Dots-and-Boxes von Übungsblatt 3 um einen KI-Gegner erweitern. Die auf Blatt 3 vorgestellten Konzepte sind weiterhin gültig. Sie dürfen als Grundlage für diese Aufgabe die Musterlösung von Übungsblatt 3 verwenden.

B.1 Grundlagen

Im Gegensatz zu Blatt 3, wo zwei Personen Dots and Boxes gegeneinander gespielt haben, soll in dieser Aufgabe ein Mensch gegen den Computer spielen. Der Mensch beginnt immer zuerst und der Computer folgt bei der Wahl seiner Züge festen Regeln.

Zuerst definieren wir, was legale und was sichere Linien sind:

Legal: Eine Linie ist erlaubt zu Ziehen, wenn diese noch nicht gezogen worden ist.

Sicher: Eine Linie ist sicher zu Ziehen, wenn diese nicht die dritte oder vierte Linie einer Box wäre. Das heißt es dürfen um diese Box vorher keine oder nur eine Linie gewesen sein.

Darüber hinaus werden Linien zuerst nach Zeile und dann nach Spalte geordnet. Das heißt die erste Linie ist folglich die ganz links in der obersten Zeile, die zweite ist rechts daneben, usw.

Der Computer wählt nun die zu ziehende Linie nach den folgenden Regeln in **genau** dieser Präzedenz:

1. Wenn der letzte Zug vom Menschen war und die um die Mitte des Feldes gespiegelte Linie legal und sicher ist, zeichne diese Linie.
2. Zeichne die erste legale und sichere Linie.
3. Vervollständige die erste Box.
4. Ziehe die erste legale Linie.

Abbildung B.2 zeigt nun einen Bespielablauf auf einem $2x2$ Feld mit amerikanischer Startbelegung. Der Mensch beginnt und gewinnt nach dem 12. Zug mit 4 zu 0 Punkten.

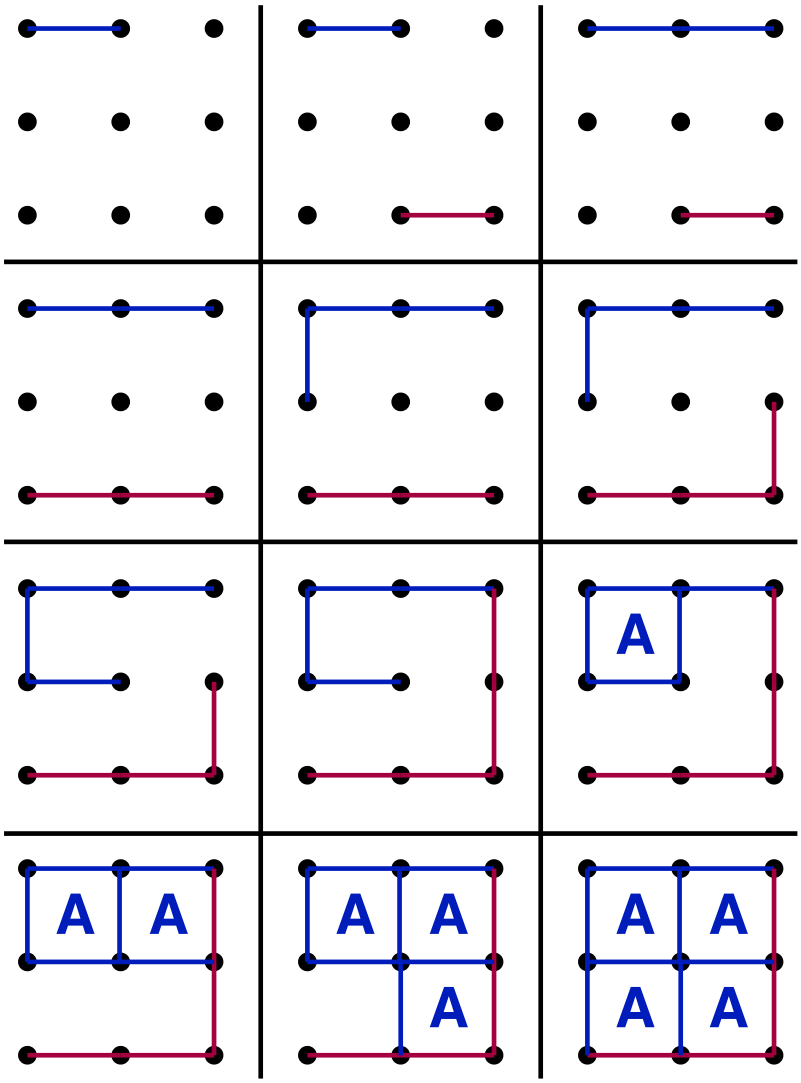


Abbildung B.2: Beispielablauf wobei der Mensch beginnt und gewinnt

B.2 Funktionalität

Im Folgenden wird die Funktionalität der Implementierung beschrieben. Es wird der Spielablauf erklärt und zudem wird erläutert, wie mit dem Programm interagiert wird.

B.2.1 Ausgabe des Spielfeldes

Die Ausgabe des Spielfeldes ist identisch zu Blatt 3, wobei der Mensch das Zeichen `A` und der Computer das Zeichen `B` hat.

B.2.2 Spielstart

Das Spiel wird ebenfalls über eine `main`-Methode gestartet. Über die Kommandozeilenargumente werden die Größe des Spielfeld (Zahl zwischen und `2` und `9`) und die Startbelegung(`american`, `icelandic`, oder `swedish`) festgelegt.

Ein Beispielaufruf ist `java DotsAndBoxes 2 american`

Geben Sie im Falle eines ungültigen Aufrufs eine aussagekräftige Fehlermeldung aus und beenden daraufhin das Programm. Fehlermeldungen müssen mit `Error ,` beginnen.

Bevor der erste Spielzug beginnt, wird das Spielfeld ausgeben. Das Programm nimmt per Kommandozeileneingabe die Züge des Menschen entgegen. Es beginnt immer zuerst der Mensch und danach folgt der Computer. Nach jedem Zug wird das Spielfeld ausgegeben.

B.2.3 Spielzug

Ein Spielzug enthält die folgenden Schritte (in dieser Reihenfolge):

1. Zu Beginn des Spielzugs des Menschen wird `"Pick a line:"` ausgeben.
2. Danach wird der Zug eingelesen und die Linie gezeichnet.
3. Achten sie im Falle einer ungültigen Eingabe auf eine aussagekräftige Fehlermeldung, woraufhin der Zug wiederholt wird. Fehlermeldungen müssen mit `Error ,` beginnen.
4. Sollte der Mensch das Kommando `quit` eingeben, so wird das Programm vorzeitig beendet.
5. Wenn der Computer am Zug ist, wählt dieser nach den oben definierten Regeln, die zu ziehende Linie.
6. Danach wird die gezogen Linie ausgegeben z.B. `"Computer chose b2 down"` und auf das Spielfeld gezeichnet.
7. Wie zuvor gilt, dass wenn jemand (Mensch oder Computer) eine Box vollständig umschließt, einen Punkt bekommt und erneut ziehen darf.

B.2.4 Spielende

Wenn alle Linie gezogen wurden, endet das Spiel. Die Person mit den meisten Boxen gewinnt. Auf der Kommandozeile wird `Human wins!` bzw. `Computer wins!` ausgegeben. Im selten Fall, dass es ein Unentschieden gibt, so wird `Draw!` ausgegeben.

B.2.5 Beispielablauf

Im Folgenden sehen sie einen Beispielauflauf des Programmes. Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Der Programmaufruf wird mit `%>` gefolgt von einem Leerzeichen eingeleitet, diese Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung zwischen Ein- und Ausgabezeilen. Ebenso ist der Programmname `DotsAdBoxes` nur beispielhaft und nicht vorgeschrieben. Nutzereingaben sind mit dem Zeichen `>` gekennzeichnet.

Beispielinteraktion

```
%> java DotsAndBoxes 2 american
   a   b
 +   +   +
1
 +   +   +
2
 +   +   +
Pick a line:
> a1 up
   a   b
 + - +   +
1
 +   +   +
2
 +   +   +
Computer chose b2 down
   a   b
 + - +   +
1
 +   +   +
2
 +   + - +
Pick a line:
> b1 up
   a   b
 + - + - +
1
 +   +   +
2
 +   + - +
Computer chose a2 down
   a   b
 + - + - +
1
 +   +   +
2
 + - + - +
Pick a line:
> a1 left
   a   b
 + - + - +
1|
 +   +   +
2
 + - + - +
Computer chose b2 right
   a   b
 + - + - +
1|
 +   +   +
```

Beispielinteraktion

```
2       |
 + - + - +
Pick a line:
> a1 down
   a   b
 + - + - +
1|
 + - +   +
2       |
 + - + - +
Computer chose b1 right
   a   b
 + - + - +
1|      |
 + - +   +
2       |
 + - + - +
Pick a line:
> a1 right
   a   b
 + - + - +
1| A |   |
 + - +   +
2       |
 + - + - +
Pick a line:
> b1 down
   a   b
 + - + - +
1| A | A |
 + - + - +
2       |
 + - + - +
Pick a line:
> b2 left
   a   b
 + - + - +
1| A | A |
 + - + - +
2   | A |
 + - + - +
Pick a line:
> a2 left
   a   b
 + - + - +
1| A | A |
 + - + - +
2| A | A |
 + - + - +
Human wins!
```