Aufgabenblatt #9

Wenn Sie es nicht bereits gemacht haben, lesen Sie bitte die relevanten Teile des Skriptums und erarbeiten Sie gemeinsam die bereits durchgegangen Fragen am Ende des Kapitels.

Siehe Skriptum Seite 402, Kapitel Grenzwerte.

Spiel

Aufgabe 1: Spielfeld

Schreiben Sie die Klasse `Spielfeld` welche eine zweidimensionale Landschaft repräsentiert. Sie soll dabei quadratisch 40x40 angeordnet sein. In jedem Feld kann ein `char` gespeichert werden. Jedes dieser Zeichen in Kleinbuchstaben repräsentiert ein Wesen:

1. m für einen Marienkäfer.
2. l für eine Laus.
3. * für den Spieler (auch eine Laus).
4. s für einen Sonnenstrahl.
5. b für ein Blatt.
6. z für ein Ziel (auch ein Blatt).

`String toString()` soll das ganze Spielfeld ausgeben. Leere Felder sollen durch ein Leerzeichen gekennzeichnet werden. Der Konstruktor soll ausschließlich leere Felder erzeugen.

Fügen Sie die Methode `add(int, int, char)` beim Spielfeld hinzu, welche es erlaubt auf einem Feld ein neues Zeichen auf leeren Feldern (angegeben durch x- und y-Koordinate) zu platzieren. Fügen Sie die Methode `equals()` beim Spielfeld hinzu um zu vergleichen ob auf zwei Spielfeldern die gleichen Zeichen vorhanden sind.

Testen Sie das Spielfeld indem Sie mehrere Zeichen darin mit `add` hinzufügen und Spielfelder mit `equals` vergleichen. Vergessen Sie nicht auf Kommentare und Zusicherungen.

Aufgabe 2: Speichern

Das gesamte Spielfeld mit Inhalt soll in eine Datei geschrieben werden können. Der Inhalt der Datei soll der Ausgabe der Methode `String toString()`

entsprechen. Schreiben Sie die Methode `toFile(String filename)` welche diese Aktion durchführt.

Erzeugen Sie Dateien in einem Testprogramm. Vergessen Sie nicht auf Kommentare und Zusicherungen.

Aufgabe 3: Laden

Implementieren Sie die Methode `fromFile(String filename)` in `Spielfeld`.

Testen Sie die Funktionalität indem Sie verschiedene Spielfelder, die zuerst herausgeschrieben wurden, dann auch wieder einlesen. Verwenden Sie die `equals` Methode um auf Gleichheit zu überprüfen.

Aufgabe 4: Bewegen

Fügen Sie die Methode `move(int, int, Direction)` hinzu. Damit geht ein Wesen (durch die Koordinaten angegeben) zu einem der benachbarten Felder (durch die Richtung angegeben). `Direction` soll dabei ein enum sein. Es ist nicht möglich ein Wesen aus dem Spielfeld heraus zu bewegen.

Sollte auf der Position bereits ein Wesen sein, verwenden Sie folgende Regeln um zu entscheiden welcher Buchstabe gewinnt:

Es also kein leeres Feld ist.

1. m (Marienkäfer) frisst l und * (Läuse)
2. l (Läuse) frisst b (Blatt)
3. * (spezielle Laus) frisst b und z (Blatt)
4. b und z (Blatt) frisst s (Sonnenstrahl)
5. s (Sonnenstrahl) verbrennt m (Marienkäfer) und l und * (Läuse)

Gibt es keine Regel, so kann der Zug nicht durchgeführt werden und das Spielfeld bleibt unverändert.

Schreiben Sie ein Testprogramm bei dem die Buchstaben auf dem Feld herumbewegt werden. Vergessen Sie nicht auf Kommentare und Zusicherungen.

Aufgabe 5: Eingaben

Schreiben Sie die Klasse `Spiel` welche in einer Endlosschleife Befehle einliest welche mit Enter abgeschickt werden. Die Befehle sollen Folgendes durchführen:

1. Bewegung von * nach links, rechts, oben oder unten.

2. Speichern und Laden des Spielfeldes, mit Dateinamen als Argument. Stellen Sie sicher dass * und z auf dem Spiel existiert, wenn ein Spiel geladen wird.

3. Beenden des Spieles.

Bevor das Spiel startet muss eine bestimmte Anzahl von Wesen auf dem Spielfeld platziert werden. Beachten Sie, dass auch genau ein Ziel vorhanden sein muss.

Platzieren Sie eine spezielle Laus (*) auf dem Spielfeld. Nur diese soll durch Sie gesteuert werden. Alle anderen Wesen auf dem Feld sollen sich dabei nach jeder Bewegung auf dem Feld zufällig herumbewegen. Frisst ein anderes Wesen Ihre Laus *, so haben Sie das Spiel verloren.

Alles außer *; also auch das Ziel soll zufällig bewegt werden.

Erreicht Ihre Laus das Zielfeld, so haben Sie gewonnen. Wie immer: Vergessen Sie nicht auf Kommentare und Zusicherungen.

Theorie

Beantworten Sie die folgenden Fragen.

Aufgabe 6: Speicher

1. Welche beiden grundlegenden Speicherbereiche werden in Java unterschieden, und welche Daten liegen in diesen Speicherbereichen?

2. Wozu dient Garbage-Collection?

3. Wie erledigt ein Garbage-Collector seine Aufgabe?

4. Welche Fallen bestehen bei Garbage-Collection?

5. Wie kann man beim Programmieren die Garbage-Collection beeinflussen?

Aufgabe 7: Dateien

1. Welche Fehler passieren leicht beim Umgang mit Dateien?

2. Was kann passieren, wenn mehrfach von derselben Datei gelesen bzw. auf dieselbe Datei geschrieben wird?

3. Was ist eine Zeichen-Codierung?

4. Wozu dienen Lock-Dateien?

Aufgabe 8: Antwortzeit

1. Was versteht man unter einer Antwortzeit?
2. Wodurch kann die Antwortzeit stärker als erwartet erhöht werden?
3. Was bedeutet Busy-Waiting?
4. Was ist schlecht an Busy-Waiting?

Aufgabe 9: Ganze Zahlen

1. Warum ist es meist keine gute Idee, ganze Zahlen durch Fließkommazahlen zu ersetzen, wenn der Wertebereich der ganzen Zahlen möglicherweise nicht ausreicht?
2. Was ist ein Überlauf oder Unterlauf?
3. Wann müssen wir `BigInteger` statt `int` oder `long` einsetzen?
4. Welche Arten von Problemen bei nicht abschätzbar großen Zahlen kann auch `BigInteger` nicht vermeiden?

Aufgabe 10: Typische Fehler

1. Welche Fehler sind Ihnen schon passiert?
2. Was sind Off-by-One-Fehler, und wodurch entstehen sie?
3. Wie kann man Off-by-One-Fehler reduzieren?
4. Welche Fallen lauern im Umgang mit `null`?