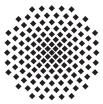


Universität Stuttgart

SQA
Software Quality and Architecture Group

Übungsblatt 06

Programmierung und Softwareentwicklung (WS 2024/2025)

Abgabe: Fr. 29.11.2024, 23:59 Uhr — Besprechung: KW 48

- Bitte lösen Sie die Übungsaufgabe in **Gruppen von 2 Studierenden**.
- Dieses Übungsblatt besteht aus zwei Teilen (A, B). Teil A ist in der Präsenzübung zu lösen. Teil B ist in Heimarbeit (Gruppe von 2 Studierenden) zu lösen und rechtzeitig abzugeben. Die Abgabe erfolgt über ILIAS.
- Geben Sie `.java`-Dateien nur im UTF-8 Encoding ab. Ändern Sie das Textdateiencoding auf UTF-8 ab, bevor Sie die Unterlagen herunterladen. Abhängig von Ihrem Betriebssystem müssen Sie möglicherweise auch nichts tun.
- Geben Sie zu Beginn der Dateien Ihre Namen (Vor- und Nachname), die Matrikelnummern und die E-Mail-Adressen an. Nutzen Sie bei Java-Dateien die korrekte JavaDoc-Syntax.
- Benennen Sie die Dateien nach dem folgenden Schema:
  1. **PSE**[**ÜB-Nr**]-[**Nachnamen der Teammitglieder**]-[**Nachname des Tutors**]**.pdf**
     Beispiel: PSE06-StießSpethKrieger-Becker.pdf
  2. [**Klassenname**]**.java**: Alle von Ihnen bearbeiteten Java-Dateien, die Lösungen für die Aufgaben enthalten.
- Missachtung der formalen Kriterien kann dazu führen, dass einzelne Aufgaben oder die gesamte Abgabe mit 0 Punkten bewertet werden.

Lernziel: Auf diesem Übungsblatt werden Sie Ihr Wissen über primitiven Datentypen, Gleichheit und Ausnahmen vertiefen. Des weiteren werden Sie den Umgang mit Kontrollflussanweisungen und das Entwerfen von Algorithmen zum lösen komplexerer Aufgaben weiter üben.

Punkte: Dieses Übungsblatt enthält zwei Teile. Im Teil B können Sie bis zu 50 Punkte und 6 Bonuspunkte erzielen. Zum Bestehen des Blatts benötigen Sie mindestens 25.0 Punkte.

Style: Bitte halten Sie die in der Vorlesung vorgestellten Style-Regeln ein. Dazu gehören auch JavaDoc sowie Vor- und Nachbedingungen. Der Style Ihrer Implementierung wird mit bis zu 50% bewertet.

Vorbereitung: Bitte erledigen Sie die folgenden Schritte **vor** der Präsenzübung.

- Importieren Sie das zu diesem Übungsblatt gehörende Maven Projekt in Ihre IDE. Sie finden das Maven Projekt in unserem git-Repository: `https://github.com/SQA-PSE-WS-2024-2025/exercise-sheet-06`
- Stellen Sie sicher, dass Sie Übungsblatt 05 absolviert haben, sowie alle Software installiert und funktionsfähig ist (IDE (Eclipse, IntelliJ, VSCode,...) und Java 21).

Unterlagen:

- Git-Repositories: `https://github.com/SQA-PSE-WS-2024-2025/`
- Dokumentation des Hamstersimulators: `https://tinyurl.com/5yx654w8`

Scheinkriterien: Durch die Teilnahme am Übungsbetrieb können Sie sich für die Teilnahme an der Klausur qualifizieren:

- Bestehen von min. 80% aller Übungsblätter.
- Ein Übungsblatt gilt als bestanden, wenn 50% der Punkte des abgegebenen Heimarbeitsteils erreicht wurden.
- Aktive Teilnahme an min. 80% der Übungen.

Viel Erfolg!

1 Teil A - Präsenzaufgaben

Aufgabe 1 Primitive Datentypen

In dieser Aufgabe setzen Sie sich mit den primitiven Datentypen in Java auseinander.

(a) Was ist der Unterschied zwischen den Typen `int` und `Integer`? Konzeptionell ist Integer die korrekte Variante, warum existiert der primitive Datentyp `int` dennoch?

(b) Viele der primitiven Datentypen repräsentieren Zahlen. Beim Rechnen mit primitiven Datentypen müssen Sie sich über deren Eigenheiten im Klaren sein. Betrachten Sie den Code-Ausschnitt in Listing 1. Überlegen Sie für die Zeilen 13 und 15 bis 19, welche Ausgaben Sie erwarten. Halten Sie Ihre Überlegungen fest. Führen Sie anschließend den Code aus. Sie finden den Code in der Klasse `PrimitiveTypes` im Paket der Präsenzteilaufgaben des Repositories.

Listing 1: Rechnen mit primitiven Datentypen

```
1  short maximumOfShort = 32767;
2
3  int tenAsInteger = 10;
4  int fourAsInteger = 4;
5  double fourAsDouble = 4;
6
7  double rootOfTwo = Math.sqrt(2);
8  double zero = (rootOfTwo * rootOfTwo) - 2;
9
10 char characterA = 'a';
11 char characterB = 'b';
12
13 System.out.println(maximumOfShort + 1);
14 maximumOfShort++;
15 System.out.println(maximumOfShort);
16 System.out.println(tenAsInteger/fourAsInteger);
17 System.out.println(tenAsInteger/fourAsDouble);
18 System.out.println(zero);
19 System.out.println(characterB - characterA);
```

Aufgabe 2 Gleichheit und tiefe Gleichheit

In dieser Aufgabe geht es um Gleichheit. Betrachten Sie dazu zuerst den Codeabschnitt in Listing 2 und beantworten Sie dann die folgenden Teilaufgaben.

Listing 2: Code

```
1  Hamster ronnie = new Hamster(game.getTerritory(), new Location(1,1), Direction.EAST, 5);
2  Hamster lisa = new Hamster(game.getTerritory(), ronnie.getLocation(), ronnie.getDirection(), 5);
```

Zu welchen Wahrheitswerten evaluieren die folgenden Ausdrücke? Notieren Sie jeweils die Wahrheitswerte und begründen Sie Ihre Antworten.

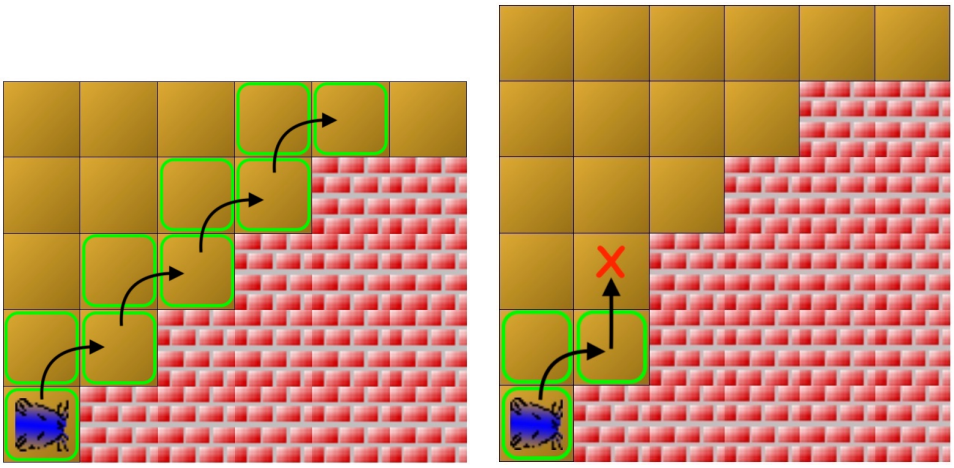
Führen Sie anschließend den Code aus und vergleichen Sie Ihre Antworten mit den tatsächlichen Ergebnissen. Sie finden den Code in der Klasse `EqualityHamsterGame` im Paket der Präsenzteilaufgaben des Repositories.

1. `ronnie.getLocation().equals(new Location(1,1))`
2. `ronnie.getLocation() == lisa.getLocation()`
3. `ronnie.getLocation() == new Location(1,1)`
4. `ronnie.getDirection() == Direction.EAST`
5. `ronnie.equals(lisa)`
6. `ronnie == lisa`

Aufgabe 3 Treppen

In dieser Aufgabe erklimmt Paule (manchmal) eine Treppe, während Sie den Umgang mit Kontrollflussstrukturen üben. Insbesondere üben Sie den Umgang mit Ausnahmen, welche Sie verwenden werden, wenn Paule daran scheitert, eine Treppenstufe zu erklimmen.

Eine Treppe besteht aus mehreren Wänden, die von links nach rechts immer höher aufeinander gestapelt sind. Eine Treppenstufe ist immer genau ein Feld breit und beliebig viele Felder hoch. Abbildung 1 zeigt Territorien mit zwei verschiedenen Treppen.



(a) Überwindbare Treppe. (b) Unüberwindbare Treppe.

Abbildung 1: Verschiedenen Treppen.

Beachten Sie:

- Paule muss beim Erklimmen einer Treppe immer auf den Stufen laufen, d.h. Sie dürfen Ihn nur über die in Abbildung 1 grün markierten Felder laufen lassen.
- Paule kann nur Stufen erklimmen, die maximal eine Wand hoch sind. An der Treppe in Abbildung 1b würde Paule scheitern.
- Gehen Sie davon aus, dass Paule zu Beginn des Spiels immer links unten sitzt.

Lösen Sie die folgenden Teilaufgaben in der Klasse `StairsPauleHamsterGame` aus dem Package `...presence.stairs`.

(a) Implementieren Sie die Operation `movePauleToNextStep`. In dieser Operation soll Paule genau eine Treppenstufe erklimmen. Falls die Stufe höher als eine Wand ist, soll eine Ausnahmen geworfen werden. Verwenden Sie dazu eine passende Exception aus dem Package `...presence.stairs.exceptions`.

(b) Implementieren Sie die Operation `hasReachedTop`, die überprüft, ob Paule am Ende der Treppe angekommen ist. Da alle Treppenstufen genau ein Feld breit sind, können Sie davon ausgehen, dass Paule das Ende der Treppe erreicht hat, sobald sich vor ihm keine weiter Stufe mehr befindet.

(c) Implementieren Sie die Operation `climbStairs`. Die Operation soll Paule von seiner Startposition links unten zum oberen Ende der Treppe bewegen. Verwenden Sie die in den vorherigen Teilaufgaben implementieren Operationen.
Beachten Sie, dass die Operation `movePauleToNextStep` eine Ausnahme wirft, falls Paule eine Treppenstufe nicht überwinden kann. Falls eine Ausnahme geworfen wird, soll der Klettervorgang abgebrochen, und eine neue Ausnahme geworfen werden. Verwenden Sie dazu eine passende Exception aus dem Package `...presence.stairs.exceptions`.

(d) Starten Sie den Hamstersimulator über die Klasse `StairsPauleHamsterGameApp` um die Korrektheit Ihrer Implementierung zu überprüfen. Überprüfen Sie Ihre Implementierung auch auf anderen Territorien.
Hinweis: Die Zeile `this.loadTerritoryFromResourceFile("/territories/StairsTerritory.ter");` im Konstruktor der Klasse `StairHamsterGame` spezifiziert, welches Territorium geladen wird. Ändern Sie `/territories/StairsTerritory.ter` zu `/territories/TooHighStairsTerritory.ter`, um ein Territorium mit einer unüberwindbaren Stufe zu laden.

(e) (Herausforderung) Schreiben Sie Ihr Programm so um, dass Paule auch Treppen erklimmt, deren Stufen breiter als eine Kachel sind.
Hinweis: Ändern Sie den an `loadTerritoryFromResourceFile()` als Argument übergebenen String zu `"/territories/TooWideStairsTerritory.ter"` um ein Territorium mit einer Stufe, die breiter als eine Kachel ist, zu laden.

2 Teil B - Heimarbeit

Aufgabe 1 Schleifen und Korrektheit (PDF)

In dieser Aufgabe ist die Operation `pickAllGrainOnPath` aus Listing 3 gegeben. Die Operation ist noch unvollständig, es fehlen Vor- und Nachbedingungen, sowie die Schleifenvariante und -invariante der Schleife. In den folgenden Teilaufgaben werden Sie sich mit möglichen Nachbedingungen, Schleifenvariante und -invariante auseinandersetzen.

Hinweis: Da dies eine Textaufgabe ist, sind die Nachbedingungen, Schleifenvariante und -invariante in den folgenden Teilaufgaben auf Deutsch formuliert. Im Code müssten Sie diese auf Englisch formulieren.

Listing 3: Eine Operation mit while-Schleife.

```
/**
 * Paule moves in a straight line up to the next wall and picks all grains on
 * the tiles he passes over.
 *
 * TODO Pre- and Postconditions in natural language
 */
void pickAllGrainOnPath() {
  /*
   * TODO Loopvariant and -invariant
   */
  while(paule.frontIsClear()) {
    if (paule.grainAvailable()) {
      paule.pickGrain();
    } else {
      paule.move();
    }
  }
}
```

(a) *(3 Punkte)* Betrachten Sie die Operation `pickAllGrainOnPath` in Listing 3. Welche der folgenden natürlichsprachlichen Ausdrücke sind korrekte und sinnvolle Nachbedingungen für die Operation `pickAllGrainOnPath`, welche nicht? Begründen Sie jeweils ihre Antwort.

i. „Paule hat alle Felder, über die er gelaufen ist, außer das direkt vor der Wand, leer gefuttert.“

ii. „Paule hat Körner im Mund.“

iii. „Alle Felder sind leer.“

(b) *(3 Bonuspunkte)* Betrachten Sie die Schleife in der Operation `pickAllGrainOnPath` in Listing 3. Welche der folgenden natürlichsprachlichen Ausdrücke sind korrekte Schleifenvarianten für die Schleife, welche nicht? Begründen Sie jeweils ihre Antwort.

i. „$anzahl_{Koerner} + anzahl_{Felder}$, wenn $anzahl_{Koerner}$ die Anzahl der bereits aufgehobenen Körner und $anzahl_{Felder}$ die Anzahl der bereits betretenen Felder ist. “

ii. „Die Summe aus der Anzahl an Feldern zwischen Paule und der in Paules Blickrichtung nächsten Wand und der Anzahl aller Körner, die auf diesen Feldern und Paules aktuellem Feld liegen.“

iii. „Die Anzahl an Felder zwischen Paule und der in Paules Blickrichtung nächsten Wand.“

(c) *(3 Bonuspunkte)* Betrachten Sie die Schleife in der Operation `pickAllGrainOnPath` in Listing 3. Beurteilen Sie, ob und wie gut die folgenden Schleifeninvarianten die in der Schleife verrichtete Arbeit beschreiben. Begründen Sie jeweils ihre Antwort.

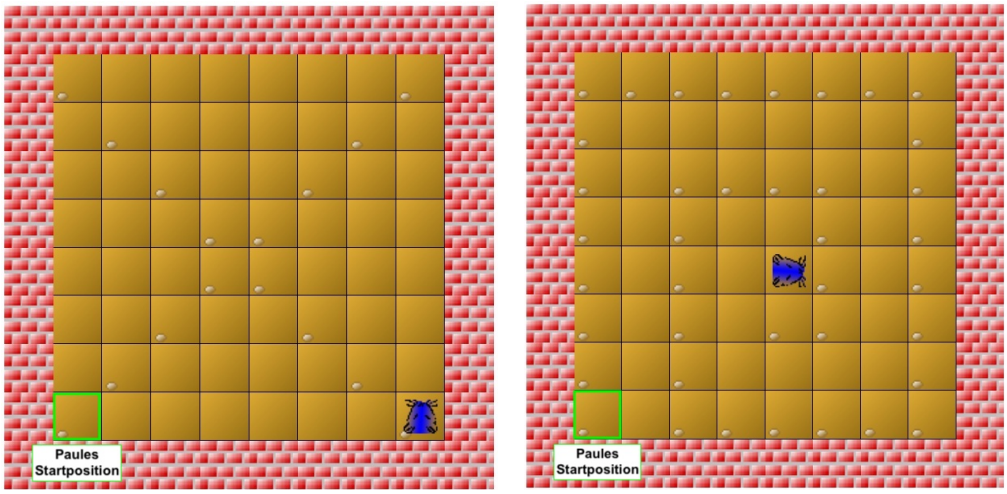
i. „Die Summe aus der Anzahl der von Paule zurück gelegten Schritte und der Anzahl der von Paule aufgehobenen Körner entspricht der Anzahl der schon ausgeführten Schleifendurchläufe.“

ii. „Auf den Feldern von Paules Startposition bis zum Feld direkt hinter Paules aktueller Position befinden sich keine Körner.“

iii. „Angenommen, i sei die Anzahl der schon ausgeführten Schleifendurchläufe, dann hat sich Paule schon i Felder bewegt und i Körner aufgehoben. “

Aufgabe 2 PainterPaule - Schleifen und Algorithmen (PDF, `.java`)

In dieser Aufgabe geht es darum, sich gorithmen zu überlegen, und diese anschließend zu implementieren. Geben Sie die Lösungen der Teilaufgaben (a) i und (b) i in der PDF ab. Geben Sie für die Lösungen der Teilaufgaben (a) ii und (b) ii `.java` Dateien ab.



(a) Territorium mit Kreuz. (b) Territorium mit Spirale.

Abbildung 2: Verschiedene Territorien aus der Vogelperspektive.

In Abbildung 2 sehen Sie Paule, der auf dem Territorium steht, von oben. Auf dem Territorium gilt es in den folgenden Teilaufgaben ein Kreuz (Abbildung 2a) beziehungsweise eine Spirale (Abbildung 2b) zu legen. Beachten Sie, dass sich bei der Spirale die „Kornlinien“ nicht berühren dürfen. Paule startet immer im linken unteren Eck mit Blickrichtung nach Norden.

(a) In dieser Teilaufgabe entwerfen und implementieren Sie einen Algorithmus, der Paule wie in Abbildung 2a abgebildet ein Kreuz legen lässt. Nehmen Sie an, dass das Territorium immer quadratisch ist und dass Paule mehr als die benötigte Anzahl an Körnern im Mund hat, d.h. auch nach dem er das Kreuz gezeichnet hat, ist Paules Mund noch nicht leer. Der Algorithmus soll für quadratische Territorien unterschiedlicher Größe funktionieren.

i. *(5 Punkte)* [.PDF] Entwickeln Sie konzeptionell einen Algorithmus, der das beschriebene Problem löst. Beschreiben Sie ihren Algorithmus in natürlicher Sprache oder Pseudocode.

ii. *(10 Punkte)* [`.java`] Implementieren Sie den in Teilaufgaben (a) i beschriebenen Algorithmus in der Operation `cross()` der Klasse `PainterPauleCrossHamsterGame`.
Hinweis: Falls Sie während dem Implementieren Fehler oder Ungenauigkeiten in ihrem Algorithmus finden und beheben, so passen Sie auch Ihre Lösung für die Teilaufgabe (a) i entsprechend an, sodass der beschriebene und der implementierte Algorithmus konsistent sind.

(b) In dieser Teilaufgabe entwerfen und implementieren Sie einen Algorithmus, der Paule wie in Abbildung 2b abgebildet eine Spirale legen lässt. Nehmen Sie an, dass das Territorium immer quadratisch ist und Paule zu Beginn genau die für die Spirale benötigte Anzahl an Körnern im Mund hat, d.h. nach dem er die Spirale gelegt hat, ist Paules Mund leer. Der Algorithmus soll für quadratische Territorien unterschiedlicher Größe funktionieren. Denken Sie an die geltenden Stilregeln, JavaDoc, Vor- und Nachbedingungen, Klasseninvarianten, Schleifen(in)varianten und so weiter.

i. *(5 Punkte)* [PDF] Entwickeln Sie konzeptionell einen Algorithmus, der das beschriebene Problem löst. Beschreiben Sie ihren Algorithmus in natürlicher Sprache oder Pseudocode.

ii. *(10 Punkte)* [`.java`] Implementieren Sie Ihren in Teilaufgabe (b) i beschriebenen Algorithmus in der Operation `spiral()` der Klasse `PainterPauleSpiralHamsterGame`.
Hinweis: Falls Sie während dem Implementieren Fehler oder Ungenauigkeiten in ihrem Algorithmus finden und beheben, so passen Sie auch Ihre Lösung für die Teilaufgabe (b) i entsprechend an, sodass der beschriebene und der implementierte Algorithmus konsistent sind.

Aufgabe 3 A'Mazing Paule - Schleifen und Algorithmen (PDF, `.java`)

In dieser Aufgabe überlegen Sie sich zunächst konzeptionell einen Algorithmus, den Sie anschließend implementieren. Geben Sie für die Teilaufgabe (a) eine PDF ab, für den Aufgabenteil (b) die vervollständigten Java-Dateien.

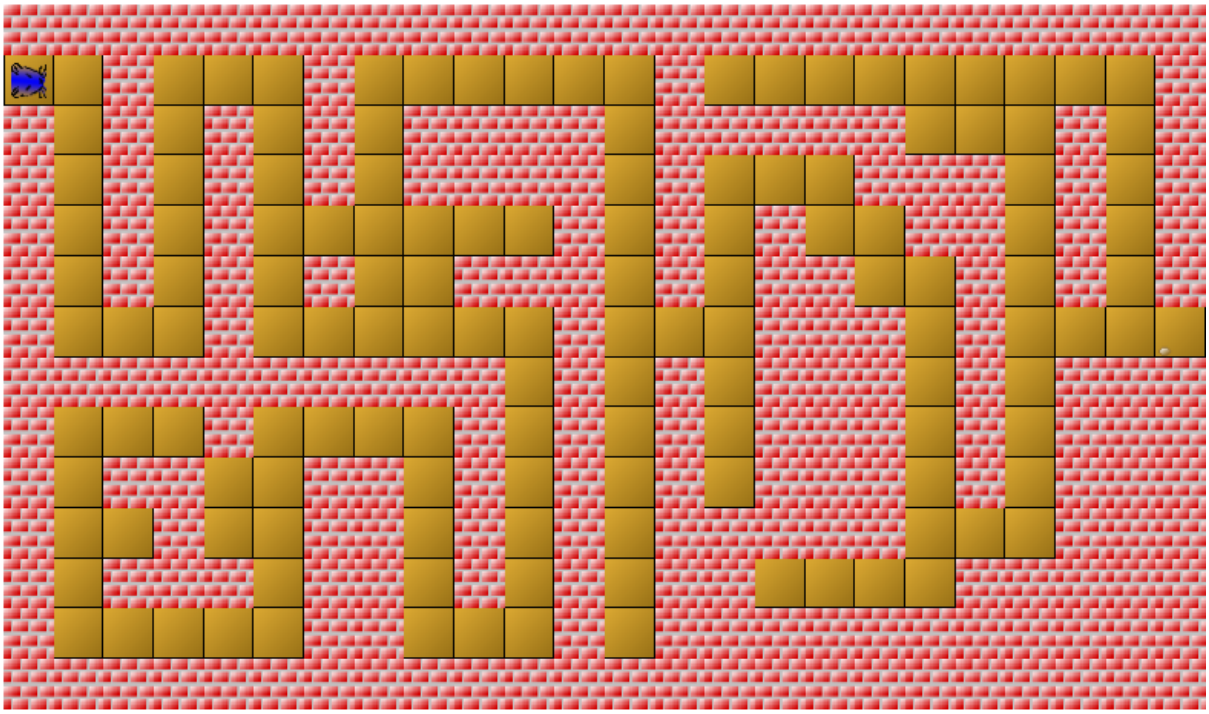


Abbildung 3: A'MazingPaule - Ein Labyrinth gilt es zu passieren.

Folgendes Szenario ist gegeben: Abbildung 3 zeigt Ihnen ein Territorium aus der Vogelperspektive, d.h. Sie sehen Paule von oben, welcher vor einem Labyrinth positioniert ist. Paule ist sehr hungrig und hat keine Körner im Mund. Am Ende des Labyrinths ist ein Korn zu finden, dass es einzusammeln gilt.

(a) *(6 Punkte)* [.PDF] Entwickeln Sie konzeptionell einen Algorithmus in natürlicher Sprache, welcher das Durchlaufen des Labyrinths beschreibt. Der Algorithmus soll dabei nicht nur spezifisch für ein bestimmtes Labyrinth funktionieren, sondern auch für anders geartete Labyrinthe. Gehen Sie davon aus, dass Paule immer am Eingang des Labyrinths positioniert ist und sich ausschließlich am Ausgang des Labyrinths ein Korn befindet, dass es einzusammeln gilt.

Hinweis: Falls Sie auf kein Ergebnis kommen, recherchieren Sie die „Linke-Hand-Regel" für Labyrinthe.

(b) *(11 Punkte)* [`.java`] Implementieren Sie Ihren Algorithmus in der Operation `passTheMaze` in der Klasse `AmazingPauleHamsterGame`. Denken Sie an die geltenden Stilregeln, JavaDoc, Vor- und Nachbedingungen, Schleifen(in)varianten und so weiter.

Hinweis: Vergrößern Sie das startende HamsterGame-Fenster, welches das Labyrinth zeigt, bis das gesamte Territorium zu sehen ist. Andernfalls kann es sein, dass Fehler geworfen werden.