



Abgabe: 21.01.2016 (vor 5:00 Uhr)

## 1 Das Projekt

Als krönender Abschluss des Praktikums soll ein kleines Spiel implementiert werden. Auf dem nachfolgenden Aufgabenblatt werden die Mindestanforderungen an die Funktionalität des Spiels erläutert. Anhand dieser Kriterien wird die Bepunktung erfolgen. Alle zusätzlich implementierten Funktionen und Erweiterungen können Bonuspunkte geben. Es können maximal 15 Bonuspunkte erreicht werden.

Wie im 1. Merkblatt angekündigt, macht das Projekt 50% der Endnote des Praktikums aus.

Das Projekt ist **einzeln** zu bearbeiten. Sie dürfen sich über Ideen mit Kommilitonen austauschen, jedoch nicht über die genaue technische Umsetzung. Es werden **Plagiatstools** eingesetzt!

Die Abgabe muss wie oben angegeben bis zum 21.01.2016 vor 05:00 Uhr erfolgen und findet wie gewohnt über Moodle statt. Die Abgabe soll in Form eines zip-Archivs (**kein** rar, 7z, tar o.ä.) erfolgen. Das Projekt muss in NetBeans ohne weitere Einstellungen außer der Einbindung des lanterna-Pakets bzw. über die Kommandozeile (mittels javac und java) ohne weitere Parameterangaben außer der Verwendung des lanterna-Pakets kompilieren und ausgeführt werden können. Es dürfen lediglich selbständig erstellte .java Dateien abgegeben werden und keine Pakete (.jar-Dateien o.ä.).

## 2 Spielidee

Ziel des Spiels ist es, eine Spielfigur durch ein Labyrinth zu navigieren. Um einen Ausgang aus dem Labyrinth zu öffnen, muss zuvor ein Schlüssel, der sich ebenfalls im Labyrinth befindet, eingesammelt werden. Zusätzlich gibt es verschiedene Hindernisse auf dem Weg durch das Labyrinth.

## 3 Mindestanforderungen

### 3.1 Spiellogik

Das zu implementierende Programm muss beliebige Labyrinth, die dem in Abschnitt 3.2 erläuterten Dateiformat entsprechen, einlesen können und diese als Spielfeld verwenden können.

Es muss ein Spielmenü geben, das vom Spieler jederzeit über die Taste *Escape* (Esc) aufgerufen werden kann. In diesem kann der Spieler auswählen, das Spiel fortzusetzen, ein gespeichertes Spiel zu laden, das Spiel zu speichern und zu beenden, das Spiel ohne zu speichern zu beenden oder sich die Legende anzuschauen. Der Inhalt der Legende wird im nächsten Abschnitt erläutert.

Der Spielfigur stehen mehrere *Leben* zur Verfügung, d.h. wenn die Spielfigur alle Leben verliert, bevor sie den Ausgang erreichen und öffnen (s.u.) konnte, hat sie verloren.

Es müssen zwei verschiedene Formen von *Hindernissen* implementiert werden, davon ist eines *statisch* und das andere *dynamisch*. Prinzipiell ist ein Hindernis eine zusätzliche Komponente auf einem ansonsten freien Weg, die bei Kontakt mit der Spielfigur dazu führt, dass die Spielfigur eine bestimmte Anzahl Leben verliert. Statisch bedeutet dabei, dass das Hindernis seine Position nicht verändert, und dynamisch, dass sich die Position des Hindernisses verändert. Die Positionen der dynamischen Hindernisse müssen regelmäßig in kurzen Abständen berechnet und ausgegeben werden.

Bevor die Spielfigur den Ausgang öffnen kann, muss sie einen Schlüssel, der sich ebenfalls im Labyrinth befindet, eingesammelt haben, d.h. an die Position des Schlüssels gelaufen sein.

Während des Spiels muss die Anzahl verbleibender Leben angezeigt werden und die Information, ob bereits ein Schlüssel eingesammelt wurde.

## 3.2 Technische Umsetzung

Das Spielfeld, d.h. das Labyrinth, soll nicht selbst erzeugt werden. Stattdessen muss ein Labyrinth aus einer Java-Properties-Datei eingelesen werden<sup>1</sup>. Dabei gilt, dass der **Key** aus der x- und y-Koordinate eines Labyrinthfeldes durch Komma getrennt besteht und der **Value** den **Typ** des Feldes angibt. Es gibt die folgenden Typen:

Value	Typ
0	Wand
1	Eingang
2	Ausgang
3	statisches Hindernis
4	dynamisches Hindernis
5	Schlüssel

Das Labyrinthfeld in der linken oberen Ecke hat die Koordinaten (0,0). Die x-Koordinaten sind nach rechts und die y-Koordinaten nach unten aufsteigend definiert. Zusätzlich gibt es die **Property-Keys** **Width** und **Height**, die als **Value** die Breite und Höhe des Labyrinths angeben. Die rechte obere Ecke des Labyrinths hat also die Koordinaten (Width-1,0), die rechte untere Ecke die Koordinaten (Width-1,Height-1) und die linke untere Ecke die Koordinaten (0,Height-1).

Achten Sie darauf, dass alle Feldtypen durch verschiedene Zeichen und Farben dargestellt werden. Die Legende im Menü soll eine Übersicht bieten, welches Zeichen für welchen Feldtypen steht.

Das Labyrinth soll in einem extra Terminal dargestellt werden. Dafür muss das *lanterna*-Paket<sup>2</sup> verwendet werden. Ein Terminal kann mithilfe des Befehls `terminal = TerminalFacade.createSwingTerminal();` erzeugt werden. Bevor Zeichen auf dem Terminal ausgegeben werden können, muss der *private Mode* aktiviert werden. Am Ende des Programs sollte dieser wieder verlassen werden.

```
terminal.enterPrivateMode();
...
terminal.exitPrivateMode();
```

<sup>1</sup>Verwenden Sie eine Suchmaschine Ihrer Wahl, um herauszufinden, wie eine Java-Properties-Datei eingelesen wird.

<sup>2</sup><https://code.google.com/p/lanterna/wiki/UsingTerminal>

Abbildung 1: Ausschnitt einer Java-Properties-Datei

```

Height=10
0,2=0
8,9=0
0,1=2
0,0=0
8,5=0
8,4=3
8,1=5
8,0=0
7,9=0
7,4=3
Width=10
7,0=1
6,9=0
6,6=4
6,0=0
5,9=0
5,8=5
5,0=0

```

Mit dem Befehl

```
terminal.moveCursor(Integer x, Integer y);
```

bewegt man den Cursor an die Position (x,y), und mit dem Befehl

```
terminal.putCharacter(character c)
```

wird an der aktuellen Position das Zeichen `c` geschrieben. Zusätzlich gibt es die Möglichkeit, die Farbe der Zeichen bzw. des Hintergrunds zu ändern, z.B. mit

```
terminal.applyForegroundColor(Terminal.Color.RED) oder
```

```
terminal.applyBackgroundColor(Terminal.Color.BLUE).
```

Aus Performancegründen sollen nur Zeichen, die sich ändern und auf dem Terminal sichtbar sind, neu geschrieben (bzw. gelöscht) werden. Mit dem Befehl

```
Key key = terminal.readInput()
```

kann die zuletzt gedrückte Taste `key` gelesen werden, und über `key.Kind()` kann abgefragt werden, um welche Taste es sich handelt. Die vordefinierten Werte können in der Dokumentation nachgelesen werden. Zur Navigation sollen die Pfeiltasten verwendet werden; diese haben die Werte `ArrowLeft`, `ArrowRight`, `ArrowUp` und `ArrowDown`.

Beachten Sie, dass das Labyrinth größer sein kann als das Terminal<sup>3</sup>. D.h. wenn die Spielfigur den Terminalrand erreicht und weitergeht (insofern das Spielfeld/Labyrinth noch weitergeht), muss ein neuer Ausschnitt des Labyrinths gezeigt werden.

Beim Speichern eines Spiels muss neben den aktuellen Positionen von Hindernissen, Schlüsseln etc. und der Spielfigurposition auch die Anzahl der verbleibenden Leben und die Eigenschaft, ob bereits ein Schlüssel eingesammelt wurde, gespeichert werden.

Das Spiel soll objektorientiert implementiert werden. Die Spielfeldtypen (Wand, Eingang, Ausgang, statisches Hindernis usw. – siehe oben) sollten alle eigene Klassen sein, die von der gleichen Oberklasse erben. Außerdem sind Codeduplikate zu vermeiden, z.B. indem

---

<sup>3</sup>Die Größe des Terminals kann mit `terminal.getTerminalSize()` abgefragt werden.

Abbildung 2: Beispiel eines kleinen Testlevels



Codefragmente sinnvoll in Methoden ausgelagert werden.

## 4 Labyrinth-Generator

Auf der Moodle-Homepage steht Ihnen ein Generator zur Erzeugung von Labyrinth, die in Form einer Java-Properties-Datei namens `level.properties` gespeichert werden, zur Verfügung. Jeder Aufruf überschreibt die Inhalte dieser Datei mit den Werten des neu generierten Labyrinths. Zusätzlich ist bereits eine Labyrinth-Datei vorgegeben, mithilfe derer Sie Ihr Programm testen können. Es müssen verschiedene Eigenschaften in einer Java-Properties-Datei namens `parameters.txt` festgelegt werden. Die Java-Properties-Datei muss beim Aufruf über die Konsole im selben Verzeichnis wie der Labyrinth-Generator liegen. Bei Verwendung von NetBeans muss sie im selben Projektordner (nicht in Unterordnern der Pakete) abgelegt werden. Folgende Eigenschaften müssen festgelegt werden:

- Breite und Höhe (**Width**, **Height**)
- Anzahl der Eingänge (**NrIn**)
- Anzahl der Ausgänge (**NrOut**)
- Anzahl der statischen Hindernisse (**StaticTraps**)
- Anzahl der dynamischen Hindernisse (**DynamicTraps**)

Das Labyrinth mit den vorgegebenen Eigenschaften wird zufällig erzeugt. Die relative Dichte des Labyrinths (Verhältnis Wände zu freien Feldern) kann über den Parameter **Density** (Default-Wert 3.0) mit beeinflusst werden, wobei die genaue Dichte dann letztlich aber immer noch zufallsbedingt ist.

## 5 Hinweise

Zum Testen Ihres Programms stehen Ihnen bereits drei verschiedene Testlevel zur Verfügung. Beim kleinen Testlevel muss die Scrollfunktion noch nicht implementiert sein. Die beiden großen Testlevel unterscheiden sich in der Dichte, d.h. in dem einen kommen viele Wände und somit schmale Wege vor, in dem anderen ist es entsprechend andersherum.

Bei zu schnellen aufeinanderfolgenden Abfragen, ob eine Taste gedrückt wurde, kann es zu

einer Überlastung und damit zu einer Verlangsamung und Störung der Zeichenausgabe im Terminal kommen. Es empfiehlt sich deshalb bei etwaigen Problemen, die stetige Wiederholung der Anweisungen mithilfe des Befehls `Thread.sleep(TimeUnit t)` zu verlangsamen.

## 6 Erweiterungen

Natürlich steht es Ihnen frei, das Spiel nach Belieben zu erweitern. Sie können zusätzliche Arten von Hindernissen, Abhängigkeiten von Schlüsseln, zusätzliche Punktesysteme (Einsammeln von zusätzlichen Elementen), weitere Funktionalitäten für die Spielfigur (Beseitigung von Hindernissen – Verschieben von Mauern, Kämpfe mit Monstern, ...), Bestenlisten usw. einführen. Achten Sie jedoch darauf, dass die Mindestanforderungen zuerst erfüllt sind **und bleiben**.

Die Verwendung anderer externer Pakete neben dem `lanterna`-Paket ist nur in Ausnahmefällen möglich. Wenn Sie ein weiteres Paket einbinden möchten, können Sie über Piazza die Erlaubnis zur Verwendung des Pakets beantragen. Dazu wird ein Link auf der Moodle-Homepage bereitgestellt werden. Geben Sie dazu den Namen, die Version, die Homepage und die Lizenz des Pakets an. Ob und welche Version des Pakets verwendet werden darf, wird dann ebenfalls über Piazza bekanntgegeben. Es wird zusätzlich die `jar`-Datei für das Paket zur Verfügung gestellt. Es werden nur Abgaben bewertet, die unter Einbindung des `lanterna`-Pakets und der weiteren über Piazza bekanntgegebenen Pakete in der entsprechenden Version kompilieren.

## 7 Dokumentation des Projekts

Der Quelltext muss aussagekräftig und prägnant kommentiert sein, so dass der Aufbau (z.B. Klassenhierarchie) und Ablauf des Programms verständlich werden. Dazu gehört insbesondere die Dokumentation der Umsetzung Ihrer statischen und dynamischen Hindernisse und der konkreten Spielregeln, die über die Mindestanforderungen hinausgehen.

Bei Erweiterungen des Projekts müssen die daraus resultierenden Spielregeln und die implementierten Zusatzfunktionen im Menü unter „Hilfe“ erläutert werden.

## 8 Checkliste

Denken Sie bei der Projekterstellung an folgende Punkte:

- Objektorientierte Umsetzung
- Einlesen des Labyrinths aus der `Java-Properties`-Datei
- Grafische Darstellung über Terminal (mithilfe `lanterna`-Paket)
- Spielinformation: Anzahl von Leben, Schlüssel anzeigen
- Menü-Aufruf (`Esc`); Menü: Speichern (Level + Spielinfo), Laden, Fortfahren, Schließen, Legende
- Scrollen
- Nur löschen/schreiben sich verändernder und im Terminalfenster befindlicher Zeichen
- Mehrere Leben ermöglichen, Interaktion mit Hindernis führt zu Verlust von Leben
- Statisches und dynamisches Hindernis, Schlüssel, Ausgang nur mit Schlüssel
- Korrekte Navigation über Pfeiltasten
- Vermeidung von Codeduplikaten – sinnvolles Auslagern in Methoden
- Angemessene und sinnvolle Kommentare/Dokumentation