TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Angewandte Informatik / Kooperative Systeme
Einführung in die Informatik I
Prof. Dr. A. Brüggemann-Klein, R. Palenta, A. Herz, Dr. A. Reuss

WS 14/15
Übungsblatt 11
10.12.2014

Abgabe: 21.01.2015 (vor 24 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 Spieles erläutert. Anhand dieser Kriterien wird die Bepunktung erfolgen. Alle zusätzlich implementierten Funktionen und Erweiterungen können Bonuspunkte geben. Es können maximal 10% der Gesamtpunktzahl des Projekts als 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 wird ein **Plagiatstool** eingesetzt!

Die Abgabe muss wie oben angegeben bis zum 21.01.2015 vor 24 Uhr erfolgen und findet wie gewohnt über den Abgabeserver als 11. Übungsblatt statt.

2 Spielidee

Ziel des Spieles 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 Labyrinthe, 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.

Dem Spieler stehen mehrere *Leben* zur Verfügung, d.h. wenn der Spieler alle Leben verliert, bevor er den Ausgang erreichen und öffnen (s.u.) konnte, hat er 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 dem Spieler dazu führt, dass der Spieler ein Leben verliert. Statisch bedeutet dabei, dass das Hindernis seine Position nicht verändert und dynamisch, dass sich die Position des Hindernisses verändert.

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

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

3.2 Technische Umsetzung

Das Spielfeld, d.h. das Labyrinth, soll nicht selbst erzeugt werden. Statt dessen muss ein Labyrinth aus einer Java-Properties-Datei eingelesen werden[1]. 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:

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. D.h. die rechte obere Ecke des Labyrinths hat 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[2] verwendet werden. Ein Terminal kann mithilfe des Befehls `terminal = TerminalFacade.createSwingTerminal();` erzeugt werden. 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.

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

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

[1] Verwenden Sie eine Suchmaschine Ihrer Wahl, um herauszufinden, wie eine Java-Properties-Datei eingelesen wird.

[2] `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
```

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, ArrowDown`.

Beachten Sie, dass das Labyrinth größer sein kann als das Terminal[3]. D.h. wenn der Spieler 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 Spielerposition auch die Anzahl der verbleibenden Leben und die Eigenschaft, ob bereits ein Schlüssel eingesammelt wurde, gespeichert werden.

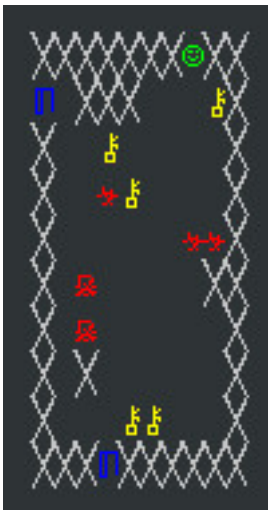
Das Spiel soll objektorientiert implementiert werden. Die Spielfeldtypen (Wand, Eingang, Ausgang, statisches Hindernis usw. - siehe oben) sollten alle eigene Objekte sein, die von der gleichen übergeordneten Klasse erben. Außerdem ist Code Duplication zu vermeiden und Codefragmente sollten sinnvoll in Methoden ausgelagert werden.

4 Labyrinth-Generator

Auf der Moodle-Homepage steht Ihnen ein Generator zur Erzeugung von Labyrinthen, die in Form einer Java-Properties-Datei namens `level.properties` (im aktuellen Verzeichnis) 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. Folgende Eigenschaften können im Labyrinth-Generator festgelegt werden.

[3]Die Größe des Terminals kann mit `terminal.getTerminalSize()` abgefragt werden.

Abbildung 2: Beispiel kleines Testlevel



a) Breite und Höhe (`Width, Height`)
b) Anzahl der Eingänge (`NrIn`)
c) Anzahl der Ausgänge (`NrOut`)
d) Anzahl der statischen Hindernisse (`StaticTraps`)
e) 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 den Spieler (Beseitigung von Hindernissen - Verschieben von Mauern, Kämpfe mit Monstern, ...), Bestenlisten, usw. einführen. Achten Sie jedoch darauf, dass die Mindestanforderungen zuerst erfüllt sind **und bleiben**.

7 Dokumentation des Projekts

Der Quelltext muss aussagekräftig und prägnant kommentiert sein, so dass der Ablauf des Programms verständlich wird.

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