Praktikumsaufgabe 6

Thema Modellierung:

6.0 Lernziele

1. Aus einer Textvorlage ein Klassenmodell entwickeln
2. Das Singleton-Pattern verstehen und anwenden.
3. Nochmals 2D Arrays – die Welt als Matrix aus Räumen.
4. Klassenmethoden schreiben.
5. Objekte aus Dateiinhalten erzeugen.
6. Das Klassenmodell refaktorisieren um Redundanzen zu beseitigen (im zweiten Teil der Aufgabe). Durch Einführen von Superklassen, Extrahieren der Gemeinsamkeiten in Methoden der Superklassen und Einführen von Hook-Methoden für den Umgang mit Abweichungen.

6.1 Allgemeine Aufgabenbeschreibung

Sie sollen das Spiel Zuul erweitern. Es sollen Bewohner und eine Reihe an speziellen Gegenständen in das Spiel eingeführt werden. Die Spielwelt soll explizit als Matrix modelliert werden, so dass wir einen guten Überblick über die Räume und die enthaltenen Objekte bekommen. Des Weiteren soll die Spielerin einen Rucksack bekommen, der in einer separaten Klasse modelliert ist.

Im nächsten Schritt, in Aufgabe 7 wird das Spiel dann um Interaktion zwischen der Spielerin, den Bewohnern und den Gegenständen erweitert.

Damit das Befüllen der Räume mit Gegenständen und Bewohnern nicht zu mühsam wird, sollen die Objekte aus Dateiinhalten rekonstruiert werden. Dazu später mehr.

6.2 Einführen der Klasse Welt für das Layout der Räume des Spiels

In dem mitgelieferten Projekt ist die Klasse ***Welt*** bereits enthalten. Mit der Welt wird auch die Klasse ***Weltanzeiger*** ausgeliefert, die sich um die Darstellung der Welt kümmert. Alles was Sie zur Darstellung tun müssen ist die Methode ***ausgeben*** auf der Welt aufrufen.

Die Welt ist in Quadranten aufgeteilt. Jeder Quadrant (eine x,y-Position in einem 2D Array) enthält genau einen Raum.

Die bereits implementierten Methoden der Klasse ***Welt***:

Methode	Funktion

`erschaffen(groesse,weltdaten_datei)`	Erzeugt die groesse x groesse Matrix für die Räume und das Objekt für die Darstellung. Merkt sich den Namen der weltdaten_datei, aus der die Objekte für die Räume gelesen werden. Ruft Methoden auf die, (1) Räume miteinander verbinden und (2) die Welt bevölkern sollen
`ausgeben()`	Stößt die Darstellung an
`name_generieren(x, y)`	Generiert einen Namen für den Raum. Der Namen Q_x_y kodiert den Quadranten des Raums. Nur diese Methode für das Generieren der Raumbeschreibung verwenden!
`raum_an_position(x, y)`	Liefert eine Referenz auf einen Raum des Quadranten x,y zurück
`position_von_raum(raum)`	Liefert die x,y Koordinaten des Quadranten des Raumes zurück.
`gueltiger_quadrant?(x,y)`	Prüft, ob der Quadrant innerhalb der Welt liegt. Wird für die Teleportation benötigt
`tunnel_vertikal(y, x_start, x_end)`	Erzeugt einen Tunnel in vertikaler Richtung in Spalte y von x_start bis y_start. Ein vertikaler Tunnel hat nur Ausgänge nach Norden und Süden. Nützlich, um die Bewegung auf dem Spielfeld einzuschränken.
`tunnel_horizontal(x, y_start, y_end)`	Analog zum vertikalen Tunnel. Hier nur Ausgänge nach Westen und Osten.
`sackgasse(x, y, zugang_aus_richtung)`	Erzeugt einen Raum mit nur einem Ausgang.
`check_in_welt(*x_y_werte)`	Prüft, ob einer der Werte nicht im Bereich der Welt liegen und wirft ggf. einen Fehler. `EndeDerWeltException`
`Welt.self.schliesse_ausgang(raum, richtung)`	Verschließt den Ausgang eines Raums in die angegebene Richtung.

TODOs in der Klasse Welt:

Aufgabe Methoden implementieren

Methode	Funktion
`raeume_erzeugen()`	Erzeugt für jeden Quadranten einen Raum und trägt diesen in die Matrix der Welt ein. Der Name des Raumes **muss** mit der Methode *name_erzeugen* generiert werden! Die Räume haben zu diesem

	Zeitpunkt noch keine Ausgänge.
***verbinden*()**	Die Methode verbindet die Räume über Ausgänge. Erlaubte Ausgänge sind die 4 Himmelsrichtungen! Nutzen Sie für die Benennung der Ausgänge die Konstanten der Klasse ***SpielUtility***. Wenn ein Raum r1 in der Matrix rechts von einem Raum r2 liegt, dann hat r1 einen Ostausgang zu r2 und r2 einen Westausgang zu r1 usw. An den Rändern darf es keine Ausgänge geben. Wir wollen ja nicht aus der Welt fallen.
***bevoelkern*()**	Liest aus der Datei ***weltdaten_datei*** (siehe Methode ***erschaffen***), die Objekte und Gegenstände, die in den Räumen der Welt platziert werden. Das Format dieser Datei und Hinweise für den Umgang mit der Datei, schließen sich an die Vorstellung der restlichen Klassen an.

6.3 Die Klasse Welt und Spiel als Singleton

Da später, wenn das Spiel interaktiv wird, Objekte anderer Klassen Zugriff auf die Welt und das Spiel benötigen, brauchen wir eine eindeutige Referenz auf das Spiel und die Welt.

Aufgabe:

Präparieren Sie die Klassen Welt und Spiel derart, dass sie sich wie Singletons verhalten! Wenn Sie keine Idee haben, wie das geht, lesen Sie das Script zu Klassenvariablen und Singletons.

6. 4 Bewohner und Gegenstände

Das Spiel bekommt die folgenden Bewohner

Name der Klasse	Parameter des initialize	Beschreibung
Monster	*name*, *lebenspunkte*, *effekt*	Monster sind feindliche Wesen, die die Spielerin angreifen können. Ein Monster hat einen Namen, eine String ohne Leerzeichen, lebenspunkte (so eine Art biologische Fitness) als ganze Zahl und

		einen Effekt. Der Effekt sind die Lebenspunkte, die die Spielerin verliert, wenn das Monster angreift.
Freund	*name*, *lebenspunkte*, *effekt*	Freunde sind altruistische Wesen, die „Leben einhauchen“. Der Effekt eines Freundes sind die Lebenspunkte, die sie verschenken. Auch hier stehen die Lebenspunkte stellvertretend für die Fitness.

Aufgabe:

Implementieren Sie die folgenden Methoden für die Klassen ***Monster*** und ***Freund***.

Methode	Funktion
`initialize(name, lebenspunkte, effekt)`	Initialisiert die Instanz-Variablen
`to_s(kurzform = false)`	Erzeugt zwei Arten von Ausgaben, die Kurz- und die Langform, je nach Wert der booleschen Variable ***kurzform***. **Beispiele für die Langform**: Ich Monster x_troll(45lp) koste Dich 30lp Ich Freund elfe_7(50lp) schenke Dir 20lp **Beispiele für die Kurzform:** M.45.30 F.50.20 Bitte exakt diese Kurzform erzeugen. Sie wird bei der Ausgabe der Welt benötigt.

Das Spiel bekommt die folgenden Gegenstände

Klasse	Parameter des initialize	Beschreibung
Waffe	*name*, *gewicht*, *effekt*, *reichweite*=*effekt*, *verbraucht_sich*= **false**	Der Effekt einer Waffe ist die Schlagkraft und wird in Lebenspunkten angegeben. Die Reichweite beschreibt, wie oft eine Waffe verwendet werden kann, bis sie unbrauchbar geworden ist. Wenn sich die Waffe verbraucht, dann wird bei jedem Einsatz der Effekt von der Reichweite abgezogen.
Zaubertrank	*name*, *gewicht*, *effekt*, *reichweite*=*effekt*, *verbraucht_sich*= **false**	Der Effekt eines Zaubertranks sind die Lebenspunkte, die die Person gewinnt, wenn sie den Trank einnimmt.

Teleporter	`name, gewicht, strecke in quadranten`	Die Strecke in Quadranten gibt an wie viele Quadranten maximal „überflogen“ werden können. Es muss nicht immer die gesamte Strecke genutzt werden. Die Strecke in Quadranten ist in einer abstrakten Lesart vergleichbar mit dem Reichweite von Waffe und Zaubertrank.
Tankstelle	`name,liter`	liter gibt den Vorrat an Treibstoff der Tankstelle an. Ein Teleporter kann an einer Tankstelle auftanken. Die Reichweite von 1 Liter bezogen auf Quadranten ist Sache der Festlegung. Die Liter der Tankstelle sind in einer abstrakten Lesart vergleichbar mit dem Reichweite bzw. der Strecke in Quadranten
Wertgegenstand	`name,gewicht,wert`	Diese Gegenstände haben eine Wert (angegeben als ganze Zahl ohne Einheit brauchen wir im Spiel zunächst nicht und macht alles komplizierter). Mit Wertgegenständen können „Körner-Sammel“-spiele u.Ä. realisiert werden.

Aufgabe:

Implementieren Sie die Methoden ***initialize*** und ***to_s(kurzform=false)*** für die Gegenstandsklassen analog zu den Bewohner-Klassen.

Beispiele für die Ausgabe

Langform	Kurzform
Waffe: xcalur(20kg) kostet 30lp Reichweite 30lp	Wa.20.30.30
Teleporter: flydsk(10kg) überwindet 20 Quadranten	Te.10.20
Zaubertrank: elixa(2kg) schenkt/kostet 10lp Reichweite 100lp	Za.2.10.100
Tankstelle: orbitfuel(Infinitykg) Vorrat: 200liter	Ta.Infinity.200
Wertgegenstand: w1(1kg) Wert 40	We.1.40

6.5 Erweiterungen: Klassen Raum Spiel Rucksack

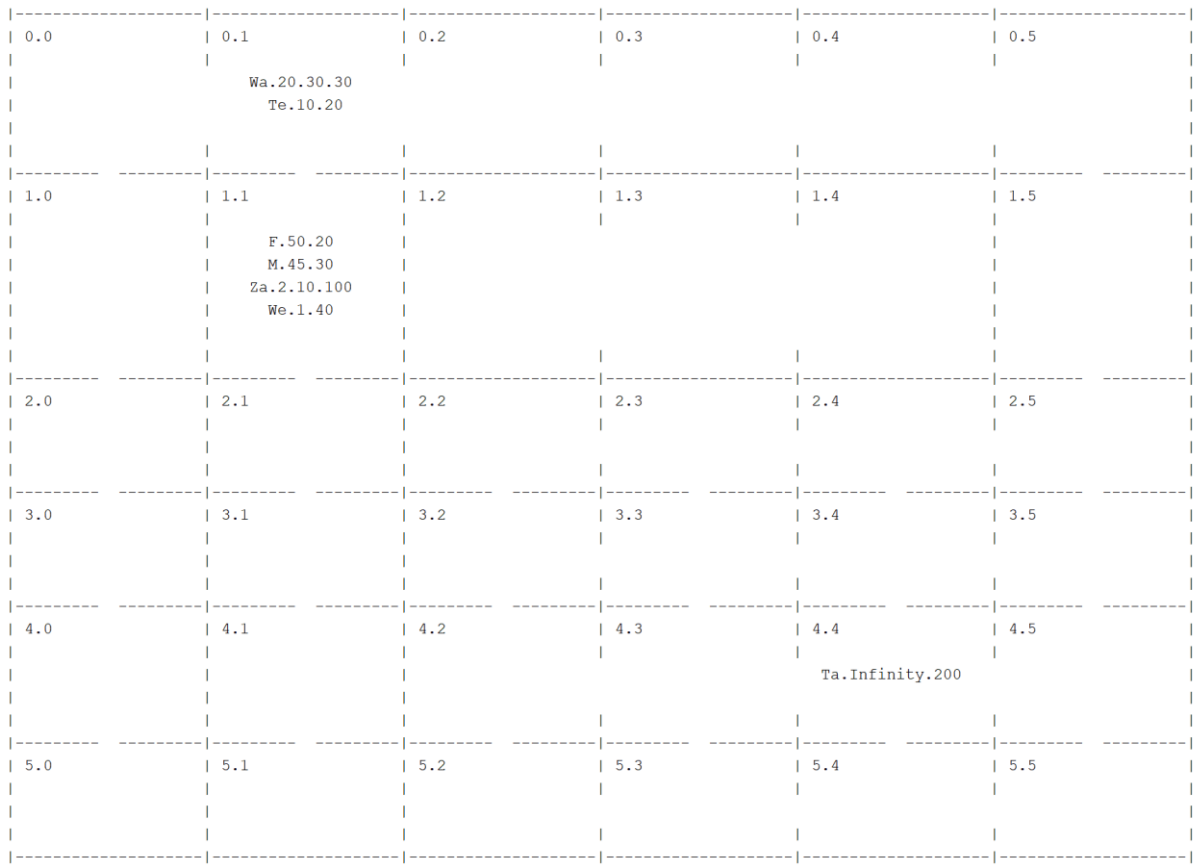
sind im Projekt enthalten

6.6 Bevölkern der Welt mit Hilfe von Dateiinhalten

Jetzt, da die Eigenschaften der Bewohner und Gegenstände erläutert sind, kann ein Format für die Darstellung der Objekte als Zeichenkette in einer Datei entwickelt

werden, die das Bevölkern der Welt vereinfacht. Dabei sollte pro Bewohner/Gegenstand auch der Quadrant des Raumes angegeben werden.

Die Datei ***weltdaten***, die die Objekte für die nachfolgende Welt beschreibt,



hat die folgenden Inhalte:

```
1 1::Freund::elfe_7::50::20
1 1::Monster::x_troll::45::30
0 1::Waffe::xcalur::20::30::30
0 1::Teleporter::flydsk::10::20
1 1::Zaubertrank::elixa::2::10::100::true
4 4::Tankstelle::orbitfuel::200
1 1::Wertgegenstand::w1::1::40
```

Die Dateneinheiten sind durch „::“ getrennt.

- Das erste Element gibt den Quadranten des Raums an. Dieses Element muss nochmals zerlegt werden und dabei in ganze Zahlen konvertiert werden.
- Das zweite ist der Name der Klasse
    - Um Zeichenketten in Klassen zu wandeln, muss die Zeichenkette zuerst in ein Symbol und dann mit ***Kernel.const_get(a_symbol)*** in das Klassenobjekt gewandelt werden.

- Das dritte und die folgenden Elemente sollen **nicht als Zeichenketten** interpretiert werden, sondern als Zahl bzw. als boolescher Wert.
    - Eine sehr einfache Art Zeichenketten in andere Typen zu konvertieren ist die Methode ***eval***. ***eval(„true“)*** liefert ***true*** , ***eval(„3“)*** liefert die Zahl ***3.*** Probieren Sie das einmal in irb aus!

- Wenn Sie ein Array ***args*** von Argumenten haben und wollen das Array in eine gültige Argumentliste umwandeln, dann müssen Sie den ***args*** beim Aufruf einen ***** voranstellen.
    - Wenn ***args = ["orbitfuel",200]*** ist, dann ist ***Tankstelle.new(*args)*** eine korrekte Parameter-Übergabe
- Mit diesen Hinweisen und dem Beispiel der Vorlesung können Sie die Weltdaten sehr generisch verarbeiten.

Aufgabe Klasse WeltdatenLeser

1. Implementieren Sie in der Klasse ***WeltdatenLeser*** die Klassenmethode ***lese_daten(dateiname)***, die die Datei mit Namen ***dateiname*** öffnet, zeilenweise liest und dabei jede Zeile in ein 2-elementiges Array umwandelt. Das erste Element des 2-elementigen Arrays ist wieder ein 2-elementiges Array für die Koordinaten des Quadranten. Das 2’te Element das Objekt (Bewohner / Gegenstand). Das Ergebnis der Methode für die oben gezeigte Weltdaten-Datei sieht wie folgt aus:

   [[[1, 1], #<Freund:0x000000000327adc8 @name="elfe_7", @lebenspunkte=50, @lebendig=true, @effekt=20>], [[1, 1], #<Monster:0x000000000327a210 @name="x_troll", @lebenspunkte=45, @lebendig=true, @effekt=30>], [[0, 1], #<Waffe:0x0000000003279400 @name="xcalur", @gewicht=20, @verbraucht=false, @effekt=30, @reichweite=30, @verbraucht_sich=false>], [[0, 1], #<Teleporter:0x0000000003278848 @name="flydsk", @gewicht=10, @verbraucht=false, @effekt=20>], [[1, 1], #<Zaubertrank:0x0000000002ca22c0 @name="elixa", @gewicht=2, @verbraucht=false, @effekt=10, @reichweite=100, @verbraucht_sich=true>], [[4, 4], #<Tankstelle:0x0000000003283e00 @name="orbitfuel", @gewicht=Infinity, @verbraucht=false, @effekt=200>], [[1, 1], #<Wertgegenstand:0x0000000003283220 @name="w1", @gewicht=1, @verbraucht=false, @wert=40>]]

2. Nutzen Sie die Ergebnisse der Methode ***lese_daten(dateiname)*** in der Methode ***bevoelkern*** der Klasse ***Welt***, um die Objekte in die Räume in den angegebenen Quadranten einzutragen. Verwenden Sie zum Eintragen der Objekte in den Raum die Methode ***<<*** der Klasse ***Raum***.
3. Testen Sie Ihre Implementierung mit der kleinen mitgelieferten Datei.

6.7 Aufgabe Entwurf einer nichttrivialen Welt

Entwerfen Sie auf dem Papier das Layout für eine Welt, in der gespielt werden soll. Legen Sie auch die Tunnel und Sackgassen fest.

Überlegen Sie, welche Bewohner und Gegenstände mit welchen Eigenschaften Sie in der Welt positionieren wollen (Vielleicht haben Sie dabei auch schon ein paar Spielregeln im Hinterkopf).

Spezifizieren Sie die dann die Weltdaten in einer Datei und legen in der Methode ***welt_erschaffen*** der Klasse ***Spiel*** die Korridore und Sackgassen für die Welt an.