hochschule mannheim

Techniken der Programmentwicklung (TPE)
Sommersemester 2014

FAKULTÄT FÜR INFORMATIK

Pflichtübung 2

Ausgabe: 14.04.2014
Abgabe: 04.05.2014 (23:50 Uhr)
Testat (IMB): 5.5.2014 (10:30–12:45 Uhr)
Testat (UIB): 9.5.2014 (15:20–17:30 Uhr)

Autor des Blattes: Jan Zieher (jan-philip.zieher@stud.hs-mannheim.de)

Aufgabe 1: Racewars **100 Punkte**

Nachdem Ihr Bankunternehmen, durch kleine Rundungsfehler zu Gunsten der Bank, beachtliche Umsätze innerhalb kürzester Zeit erzielt hat, entlohnt Sie diese zusätzlich mit einer Auszeichnung zum Entwickler des Jahres im Bereich „Top Financial Products“. Diese Anerkennung machte ebenfalls in der Spielbranche die Runde, weshalb Sie nun ein Angebot des Spielgiganten Blizzard auf Ihrem Schreibtisch liegen haben, ein neues rundenbasiertes Rollenspiel zu entwerfen. Da Sie selbst leidenschaftlicher Zocker sind, nehmen Sie diese Herausforderung natürlich sofort dankend an. Die Anforderungen an das Spiel werden nun im Folgenden näher erläutert:

(a) **Beschreibung des Spiels**

Bei dem Spiel geht es um den Kampf zwischen unterschiedlichen Wesen aus einer dunklen Fantasywelt. In dem Spiel gibt es vier verschiedene Rassen zur Auswahl.

- **Orks** – Da wären zum Einen die Orks, die die Eigenschaft besitzen sehr stark zu sein. Ein besonderer Ork ist der Held **Farseer**, der die Ork-Horden anführt.
- **Untote** – Weiter soll der Spieler die Möglichkeit haben Untote als Rasse auszuwählen. Diese sind sehr günstig und nur in der Masse stark. Der Anführer der Untoten ist der dunkle **Lich**.
- **Menschen** – Des Weiteren gibt es die Menschen. Diese sind sehr ausdauernd, was bedeutet, dass sie größeren Schaden als andere Wesen einstecken können. Sie regiert der **Erzmagier** und führt sein Gefolge in die Schlacht.
- **Nachtelfen** – Zu guter Letzt gibt es noch die Nachtelfen, welche die Eigenschaft besitzen sehr intelligent zu sein. Der mächtigste Elf ist der **Dämonenjäger**.

Alle Rassen haben wichtige Gemeinsamkeiten: Dazu gehören **Lebenspunkte**, **Rüstung**, **Schaden**, **Geschwindigkeit**.

Anführer besitzen zusätzlich ein **beherrschtes Element**. Die Elemente sind **Feuer**, **Wasser**, **Luft** und **Erde**. Weitere Elemente gibt es nicht. Wählen Sie daher für die Elemente eine adäquate Darstellung. Die Zuordnung zwischen Anführer und Element ist:

Anführer	**Element**
Farseer	Erde
Lich	Wasser
Erzmagier	Feuer
Dämonenjäger	Luft

Tabelle: Elemente der Anführer

Von den Anführern der jeweiligen Rassen (Farseer, Lich, Erzmagier und Dämonnejäger) darf es immer nur ein einziges Objekt geben. Ergreifen Sie entsprechende Maßnahmen, um dies sicherzustellen.

Die Instanzen der einzelnen Wesen werden über eine Factory (**WesenFactory**) erzeugt. Eine direkte Erzeugung ist nicht möglich. Der Factory übergibt man die Information, wie viel Geld man investieren möchte und sie gibt ein Array von Wesen zurück. Das erste Element ist immer der Anführer, die folgenden

dann entsprechend die Wesen der Art. Reicht das Geld nicht, wird ein leeres Array zurückgegeben. Die Art der Erzeugten Wesen wählt man über eine Enumeration (**Rasse**) aus, die auch die Kosten (s. u,) pro Art kennt.

Art	**Kosten**
Ork	150
Farseer	300
Untote	70
Lich	140
Mensch	110
Erzmagier	220
Nachtelf	145
Dämonenjäger	290

Tabelle: Kosten für die Wesen

Vor Spielstart müssen sich zwei Spieler einen **Squad** zusammenstellen, welches Wesen aus maximal zwei unterschiedlichen Rassen enthält. In einem Squad können beliebig viele Wesen der jeweiligen Rasse vorhanden sein, der Anführer aber nur einmal. Die Anzahl der Wesen wird durch deren Kosten begrenzt. Jeder Spiele hat einen festen Geldbetrag von 2000 Elfendollar. Mit diesem kann er Wesen für seinen Squad kaufen.

Der Spieler gibt seinem Squad neben den Wesen noch die Information mit, wie es heißen soll. Diesen Namen kann er frei wählen (z. B. „The walking dead“).

Allen Wesen gemeinsam sind die Eigenschaften Lebenspunkte, Schaden, Geschwindigkeit, Rüstung und Spezialattribut. Diese sind wie folgt belegt:

Art	**Lebenspunkte**	**Schaden**	**Geschwindigkeit**	**Rüstung**	**Spezialattribut**
Ork	100	33	1	30%	4,0
Untote	120	16	2	30%	1,6
Mensch	140	40	2	40%	10,0
Nachtelf	120	15	3	20%	2,9

Tabelle: Fähigkeitswerte der Wesen

Die Anführer besitzen zusätzlich einen **Bonus Faktor**. Dieser wird auch als Multiplikator für die Heldenlebenspunkte verwendet.

Anführer	**Bonus**
Farseer	1,2
Lich	2,3
Erzmagier	5,0
Dämonenjäger	3,0

Tabelle: Bonus der Anführer

Jede Rasse verfügt zusätzlich über die Schnittstelle **Kaempfer**, damit jedes Objekt über die Funktion `attack(Kaempfer r)` verfügt, sodass ein Angriff auf einen anderen Kämpfer ausgeführt werden kann. Diese Methode gibt den angerichteten Schaden zurück und zieht gleichzeitig diesen von den Lebenspunkten des Gegners ab.

Jede Rasse verfügt zudem über die gleichen Methoden `isLebendig()`, die angibt, ob ein Wesen noch lebt. Ist ein Wesen nicht mehr lebendig, wird es aus dem Squad entfernt.

Weiterhin gibt es eine `beschraenkeSchaden(double damage)`. Bei Menschen führt diese Methode dazu, dass der angerichtete Schaden prozentual um den Wert des Spezialattributes verringert wird. Bei anderen Wesen hat sie keine Wirkung.

Ein weiteres wichtiges Kriterium ist, dass die spezifischen Eigenschaften der einzelnen Rassen unveränderbar nach Objekterstellung sind und von der Hauptklasse **Wesen** generell kein Objekt erstellt werden darf.

Neben den dargestellten Rassen gibt es noch **Geister**. Diese haben keine der beschriebenen Eigenschaften oder Methoden, können aber trotzdem über `attack(Kaempfer r)` an Kämpfen teilnehmen. Da sie

nur ätherische Wesen sind, kann man sie nicht töten, sie ziehen dem Gegner aber grundsätzlich einen Lebenspunkt ab. Man kann Geister nicht in sein Squad aufnehmen, sondern sie werden vom Spielsystem in den Kampf geschickt.

Nun zur generellen Kampflogik des Spiels. Jede Schadenshöhe einer Rasse berechnet sich wie folgt:

$damage = Geschwindigkeit \cdot Schaden \cdot Spezialattribut$

wobei letzterer bei den Menschen wie schon erwähnt beschränkt wird, da diese mehr Schaden ertragen als andere Klassen. Achten Sie daher darauf den errechneten Schaden bei den Menschen um diesen Faktor zu reduzieren. Jeder Anführer erhöht diesen Schaden nun noch einmal um seinen jeweiligen Bonus Faktor.

$damage = damage \cdot bonus$

Sollte der Anführer auf einen anderen Anführer treffen und dessen Element überlegen sein, wird der Schaden noch einmal verdoppelt. Hierbei gilt: Feuer ist Luft überlegen, Wasser ist Feuer überlegen, Erde ist Wasser überlegen, Luft ist Erde überlegen.

Nachdem der Schaden errechnet wurde muss noch wie oben angedeutet die Rüstung beachtet werden. Das heißt der angerichtete Schaden muss prozentual der angegebenen Rüstung reduziert und anschließend von den Lebenspunkten abgezogen werden.

Jeder Held verfügt zusätzlich über eine Spezialfunktion die nur jede 3. Runde aufgerufen werden kann.

- Dämonenjäger – `goldschuss(Squad s)` zieht allen Einheiten des Squads 25 ihrer Lebenspunkte ab
- Erzmagier – `absorption()` Bekommt eine Runde keinen Schaden
- Farseer – `doppelschlag(Sqaud s)` löscht 2 zufällige Wesen des Gegners komplett aus
- Lich – `verwesung(Squad s)` entzieht allen Einheiten des Squads 7 Punkte und gibt sich selbst die entzogenen Lebenspunkte

Die Spezialfunktionen sollen Sie implementieren, es ist aber nicht nötig, die Methoden während der Simulation (s. u.) zu rufen.

Der **GameController** erstellt die Squads und besitzt die Kenntnis über die aktuelle Runde sowie den Spieler der an der Reihe ist. Außerdem startet seine Funktion `runGame()` das Spiel. In dieser wird in einer Schleife jeweils die Attacke eines Squad auf das andere ausgeführt, und jeweils der aktuelle Spielstand sowie Runde über die statische Methode `printGame(GameController game)` des **GameViewers** ausgegeben. Jedes Wesen eines Squads führt eine Attacke auf eine zufällig gewählte Wesen des Gegner Squads durch. Wenn das Spiel gestartet wurde läuft es ohne Nutzereingaben bis zur kompletten Auslöschung eines Squads und terminiert mit der Ausgabe des Gewinners. Bei jedem Kampf werden die Ergebnisse der Begegnug und die Auswirkung auf den Squad angezeigt.

Die Geister brauchen Sie in der Simulation nicht vorzusehen. Es reicht, die Klasse anzulegen.

Überschreiben Sie die `toString`-Methoden so, dass sinnvolle Informationen zu den einzelnen Wesen ausgegeben werden können.

(b) **UML-Diagramm**

Erstellen Sie ein Klassendiagramm, das die im Text genannten Entitäten, Eigenschaften und Methoden in einem objektorientierten Modell abbildet. Lesen Sie den Text sorgfältig, um keine Klassen oder Interfaces bzw. Beziehungen zu vergessen.

Nutzen Sie Vererbung, abstrakte Klassen und Interfaces, um keine Information und kein Verhalten mehr als einmal abbilden zu müssen. Sie werden höchstwahrscheinlich noch eigene, über die im Text genannten Entitäten herausgehende Abstraktionen einführen wollen, um das Design zu vervollständigen. Teilen Sie die entworfenen Strukturen in verschiedene Pakete auf, und wählen Sie sinnvolle Sichtbarkeitsattribute.

(c) **Implementierung**

Implementieren Sie alle Klassen aus Ihrem UML-Diagramm mit dem entsprechenden Verhalten.

(d) **JUnit**

Implementieren Sie sinnvolle JUnit Tests zur Überprüfung der Singletons, der Kampf-Funktion und der Erstellung verschiedener Squads

(e) **Simulation**

Simulieren Sie eine Schlacht mit einer gewissen Anzahl an Einheiten, jedoch aber gleichem Startkapital beider Squads. Machen Sie entsprechende Ausgaben, aus denen hervorgeht, dass sich Ihr Programm wie spezifiziert verhält.

Achtung
Bitte beachten Sie folgendes, damit es nicht zu unnötigen Punktabzügen in der Bewertung kommt:

- Benennen Sie Klassen und Methoden konsistent und verständlich. Klassen sollten Nomen als Namen, Methoden Verben haben.
- Dokumentieren Sie alle Klassen, Interfaces, Konstruktoren und Methoden mit JavaDoc. Dokumentieren Sie auch alle Parameter, Rückgabewerte und Ausnahmen.
- Formatieren Sie Ihren Code konsistent. Ein guter Standard sind 4 Spaces Einrückung pro Ebene ohne Verwendung von Tabulatoren.
- Halten Sie sich an die Sun Java-Code-Convention
  http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Programmieren Sie defensiv und testen Sie Eingabewerte auf deren Gültigkeit. Ihr Programm darf auch mit Daten nicht abstürzen, die außerhalb der Aufgabenstellung liegen.
- Machen Sie keine Konsoleneingaben oder -Ausgaben, es sei denn die Aufgabe fordert dies explizit.
- Halten Sie Daten und Methoden zusammen. Trennen Sie diese nicht unnötig auf.
- Kopieren Sie keinen Code sondern versuchen Sie mit den bekannten Mitteln der Objektorientierung Code-Duplikate zu vermeiden.