Universität Hamburg
Department Informatik
Students, SSE

Ein KUULer Praktikumsbericht

Praktikumsbericht

Sofrwareentwicklungspraktikum
Agile Softwareentwicklung

Louis Kobras, Utz Pöhlmann
Matr.Nr. 6658699, 6663579
4kobras@informatik.uni-hamburg.de, 4poehlma@informatik.uni-hamburg.de

28. Oktober 2015

Inhaltsverzeichnis

1 KUUL?

Dieser Bericht befasst sich mit dem Praktikum Agile Softwareentwicklung[1] (im Folgenden „ASE“) als Teil des Moduls Softwareentwicklungspraktikum[2] (im Folgenden „SEP“) , absolviert im zweiten Fachsemester des Studiengangs Software-System-Entwicklung .
Dieser Bericht ist geschrieben aus Sicht der Teilnehmer des Teilkurses 3 (ASE-SEP3). In diesem wurde das Zuul-Ausgangssystem, welches von den Praktikumsbetreuern bereit gestellt wurde, zu einem *Point & Click*[3]-ähnlichen *Adventure-Game*[4] mit dem Titel *KUUL* weiterentwickelt.
In diesem Bericht wird tiefergehend auf die Implementation von interaktiven Objekten in das *KUUL*-System eingegangen. Dies umfasst die Erstellung der Gegenstände, deren Speicherung und die Implementation der Interaktion.

2 Das Projekt

Wie bereits in der Einleitung angesprochen, wird sich dieser Bericht primär mit dem Umgang mit den Gegenständen in *KUUL* befassen. Interaktiv sind diese Gegenstände insofern, als dass sie aufgesammelt, benutzt und kombiniert werden können.

2.1 Organisation im Team

Zunächst ist die Organisation innerhalb des Teams zu besprechen. Da es sich bei dem Praktikum um einen Einblick in Agile Software-Entwicklung handelt, wurden agile Methoden wie das *Stand-Up Meeting* oder das *Kanban-Board* verwendet.
Es wurde in Zweierteams gearbeitet, die - mehr oder weniger - täglich rotiert sind, damit sich das Wissen um den Code möglichst gleichmäßig im Team verbreitet. Beim *Stand-Up Meeting* handelt es sich um einen „Stehkreis“, in dem jeder Mitarbeiter kurz erklärt, was er seit dem letzten *Stand-Up Meeting* gemacht hat. Diese wurden ziemlich schnell in regelmäßigen Abständen Zwei bis Vier mal am Tag abgehalten. Das *Kanban-Board* ist eine Tafel, an die die aktuellen Aufgaben angeschrieben werden. Dies hilft, die Aufgaben aufzuteilen und nicht den Überblick zu verlieren. Zudem ist es ein Ort für dringende Aufgaben oder Probleme, damit das Team, welches als nächstes mit seiner Aufgabe fertig wird, sofort eine neue findet. Eine weitere agile Methode ist *Schätzpoker*. Hier wird nach jeder Aufgabe, die das Team vom Kunden erhält, geschätzt, wie umfangreich diese wohl in der Umsetzung sein wird. Dies wird auch am *Kanban-Board* vermerkt, um besser planen zu

[1] STiNE-Veranstaltungsnummer 64-142
[2] STiNE-Modulnummer SSE_PR
[3] Ein Programm, bei dem Aktionen durch ein Anvisieren mit einem Zeiger (meist die Maus) und das darauffolgende Klicken hervorgerufen werden.
[4] Ein Spiel, bei dem die Erkundung und das Lösen von Rätseln im Vordergrund stehen.

können.
Agile Methoden im Allgemeinen dienen dazu, bei möglichst flexibler Organisation möglichst effizient das Projekt voranzubringen. Hierfür war stete Zwei-Wege-Kommunikation mit dem Kunden erforderlich, was insofern geschah, als dass der aktuelle Projektstand sowie Änderungswünsche und Anregungen des Kunden bis zu drei mal täglich besprochen wurden. Der Kunde ist diejenige Person, die den Auftrag aufgibt und das Produkt entgegen nimmt, was in unserem Fall die Projektleiter einer anderen Gruppe - namentlich Fred und Michael - waren.
Im Gegensatz zum veralteten Wasserfall-Verfahren bietet die agile Entwicklung den Vorteil, dass das Projekt stetig in der Entwicklung beobachtet werden kann und offen für spontane Änderungen ist. Das *Kanban-Board* dient hierbei zur Übersicht und Aufteilung des Projektes und das *Stand-Up Meeting* zur schnellen und informativen teaminternen Kommunikation.
Nun stellt sich jedoch die Frage, um welches Projekt es sich handelt. Es handelt sich um *KUUL*.

2.2 Spielprinzip und Hintergrund von *KUUL*

KUUL ist ein Ausbau des von den Praktikumsbetreuern bereitgestellten Zuul-Ausgangssystems.
In *KUUL* landet der Spieler in einer Welt und muss sich dort zurecht finden. Die Welt von Kuul besteht aus mehreren Räumen, durch die und in denen man sich bewegen kann. In diesen Räumen gibt es Objekte, mit denen der Spieler interagieren kann und so muss er nach und nach die Welt erkunden und einige Rätsel lösen, um herauszufinden, wie er das Spiel gewinnen kann.
Während Zuul ein konsolenbasiertes Text-Adventure ist, wird *KUUL* mit Maus und Tastatur gesteuert. Dies geschieht insofern, als dass Gegenstände, *NPCs*[1] und Türen angeklickt werden können, um mit ihnen zu interagieren. Mithilfe der Tastatur kann der *PC*[2] durch die Räume in der Welt von *KUUL* bewegt werden, ebenso lassen sich alle Funktionen des Spiels über bestimmte Tasten direkt ansteuern.
Zuul hatte bereits einige wenige Räume vorzuweisen, die durch die Namensgebung erkennen ließen, dass es sich um eine an ein Universitätsgelände (im Folgenden „Universität“) angelehnte Welt handelte. Dies wurde vom Entwicklerteam aufgegriffen, wobei die Umgebung *Universität* zwar beibehalten, jedoch die Anzahl der Räume mehr als verdoppelt wurde. Auf Wunsch des Kunden hin wurde im Lauf des Praktikums eine weitere Welt hinzugefügt, die sich grob am Hamburger Kiez (im Folgenden „Kiez“) orientiert.

2.3 Funktionsumfang

Wie bereits angesprochen, können Gegenstände aufgehoben und genutzt werden. In Zuul ging dies über die Konsole. Um *KUUL* komfortabler zu machen, sind Ge-

[1] *Non-Player-Character*, eine Figur im Spiel, die nicht vom Spieler, sondern vom Spiel kontrolliert wird.
[2] *Player-Character*, vgl. *NPC*

Abbildung 1: Alle Gegenstände aus der Universität

genstände im Raum anklickbar. Ebenso können sie aufgehoben werden, wenn sich der Spieler direkt davor stellt und die Enter-Taste drückt.
Aufgenommene Gegenstände kommen in den *Rucksack*, welcher allerdings nur eine beschränkte Kapazität von vier beliebigen Gegenständen hat. Gegenstände, die man im Rucksack hat, können kombiniert werden. Allerdings müssen sie dazu in der XML-Datei den Tag „kombinierbar“, welcher im Parser als `boolean` ausgelesen wird, gesetzt bekommen. Ist dieser Tag auf den Wert `„false“` gesetzt, so können die Gegenstände nicht kombiniert werden.
Eine weitere Besonderheit des Rucksacks ist es, dass die meisten Gegenstände in der ersten Welt, die einem normalerweise Energie zurückgeben, mit der Zeit verfallen. Energie wird benötigt, um sich zu bewegen. Der Spieler startet mit 15 Energie-Einheiten und jeder Raumwechsel verbraucht eine Einheit. Das Verfallen führt dazu, dass sie irgendwann keine Energie mehr zurückgeben, sondern sogar welche abziehen.
Illustriert wird dies erstens durch eine Veränderung des Aussehens des Gegenstandes und zweitens durch eine Änderung des Textes, der bei Nutzung des Gegenstandes angezeigt wird. So wird z.B. *„Die Ananas stärkt Sie und gibt Ihnen Kraft. Sie erhalten 2 Lebenspunkte. Toll!“* zu *„Die Ananas ist bis auf den Kern vergammelt.*

Sie verlieren Lebenspunkte!“, wenn man die Ananas lange genug im Rucksack behält.
Graphisch hat jeder Gegenstand eine bis drei Phasen, der Abzug der Energie-Einheiten wächst allerdings stetig linear. Alle zwei Schritte zieht der Gebrauch eines solchen Gegenstandes eine Einheit mehr ab.

```
1  if (_verfallszeitpunkt > 1) {
2      _verfallszeitpunkt = 0;
3      lebensCommand.verringereLebenspunkte();
4      if (lebensCommand.getLebensEffekt() == 0) {
5          setIcon("Graphics/Items/Phase2" + "/" + gibName() +
               "2.png");
6          _verfallStatus = 1;
7          return true;
8      } else if (lebensCommand.getLebensEffekt() < 0) {
9          setIcon("Graphics/Items/Phase3" + "/" + gibName() +
               "3.png");
10         _verfallStatus = 2;
11         return true;
12     }
13 } else {
14     _verfallszeitpunkt++;
15 }
```

Code 1: Teilimplementation der Verringerung der Energieregeneration

Bei dem Feld `_verfallszeitpunkt`[1] (Code 1) handelt es sich um eine Exemplarvariable vom Typ Integer. Immer dann, wenn Code 1 aufgerufen wird, wird diese Variable geprüft. Ist sie auf 0 oder 1, wird sie inkrementiert; ist sie auf 2, so wird der Effekt des Lebenskommandos[2] dekrementiert und der `_verfallszeitpunkt` wird wieder auf 0 gesetzt. Was ebenfalls zu sehen ist, ist, dass ein neues Icon für den Gegenstand gesetzt wird, wenn er 0 Energie-Einheiten erreicht, beziehungsweise sobald er anfängt, Energie abzuziehen (s. Code 1 Zeile 4-12).

Zur bereits angesprochenen Kombinationsfunktion sei gesagt, dass nicht alle Gegenstände kombinierbar sind. Es wurde bereits gesagt, dass ein Tag gesetzt werden muss, damit ein Gegenstand vom *Crafting Tool* (vgl. Abb. 2) akzeptiert wird. Desweiteren muss es ein Rezept geben, welches den fraglichen Gegenstand beinhaltet. Befindet sich dann der zweite Gegenstand ebenfalls im Rucksack des Spielers, so kann der Spieler die Kombination ausführen.
Die hier betrachtete Konfiguration des *Crafting Tools* entstammt der ersten Welt. Das Messer und das Gemälde werden kombiniert, um einen Schokoriegel zu erhalten, welcher die Energie vollständig wieder auffüllt. Messer und Gemälde werden dabei im Inventar durch den Schokoriegel ersetzt.

[1] Nomenklaturregeln nach SE2-Format
[2] interne Umsetzung der Energieverwaltung, für die Verwendung der Gegenstände nicht weiter von Bedeutung. Für Kommandos s. 2.5

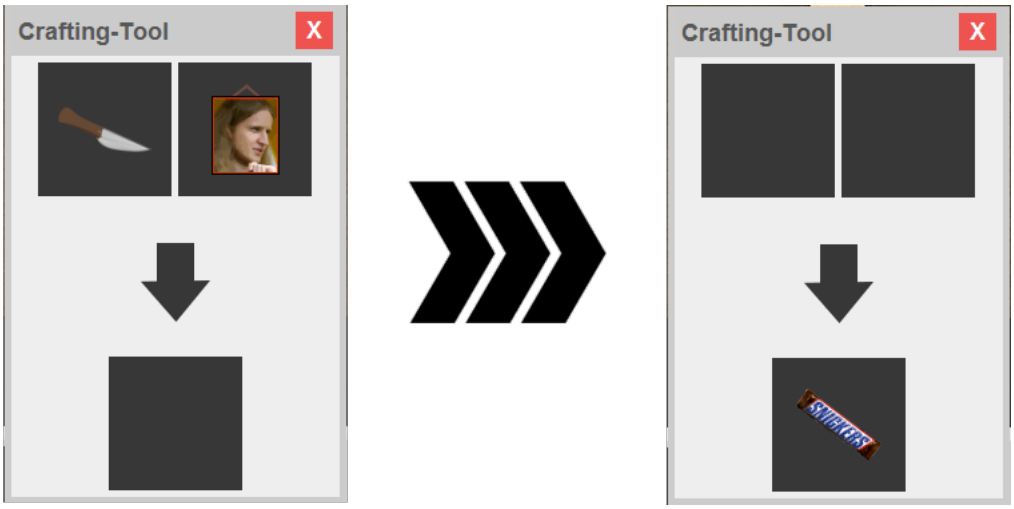

Abbildung 2: Crafting Tool

2.4 Ein Spiel beenden

Da zunächst nur die Textwelt von Zuul zur Verfügung stand, wurden die ersten Gegenstände inklusive des provisorischen Gewinngegenstandes *Handy* in den Räumen platziert, die nach der Erweiterung vorhanden waren. In *KUUL*, ist es auf zwei Arten möglich, zu gewinnen: Entweder findet der Spieler einen Schlüsselgegenstand, mit dem er einen Raum betreten kann, der als *Siegraum* betitelt wurde, oder er findet einen *Sieggegenstand*. Auf Kundenwunsch wurde das Handy als Sieggegenstand recht schnell von der *Goldananas* abgelöst.

```
1 <beenden>Du hast die goldene Ananas gegessen! Toll!</
    beenden>
```

Code 2: XML-Tag für den Sieggegenstand

```
1 if (gegenstandElement.getElementsByTagName("beenden").
    getLength() != 0)
2 {
3     Node beendenNode = gegenstandElement.
          getElementsByTagName(
4               "beenden").item(0);
5     String nachricht = beendenNode.getTextContent();
6     _beendenCommandListe.put(id, nachricht);
7 }
```

Code 3: Entsprechender Auszug aus dem XML-Parser für Gegenstände

Der hier betrachtete Quellcode (Code 3) ist aus dem in *KUUL* verwendeten XML-Parser entnommen. Er sucht sich den Tag `beenden` (dargestellt in Code 2) und packt den zum Tag gehörigen Gegenstand zusammen mit der Nachricht, die sich in dem Tag befindet, in eine Liste. Diese Liste kann nun an anderer Stelle durch-

laufen werden, um die Kommandos[1] festzulegen.
Die XML[2]-Sprache arbeitet insofern, als dass jedes Dokument eine Reihe von Elementen hat, und diese Elemente haben Attribute (im Folgenden „Tags“), welche wiederum Element-Reihen seien können. Im Gegensatz zu HTML[3] ist XML erweiterbar, das heißt es können benutzerdefinierte Tags hinzugefügt werden; die Syntax der beiden Sprachen ist nahezu gleich.
An dieser Stelle wird der Tag des Gegenstands mit der Bezeichnung „beenden“ abgefragt: Ist sie vorhanden (Länge ungleich 0), so handelt es sich um einen Sieggegenstand. Anzumerken ist hierbei, dass der verwendete Parser abgefragte Tags immer als eine Form von Liste zurückgibt, weswegen der Tag mit `.item(0);` abgefragt werden muss.
Die Siegnachricht, welche ebenfalls extern in der XML-Datei gelagert wird, wird ausgelesen und als String gespeichert. Der Gegenstand wird als Tupel aus ID und der Siegnachricht in eine Liste gelegt, die an anderen Stellen im Programm aufgerufen werden kann, um festzustellen, ob der gerade verwendete Gegenstand ein Sieggegenstand war.
Wird der Sieggegenstand nun innerhalb des Spiels gefunden, aufgehoben und benutzt, so wird das Spiel beendet und es erscheint ein Popup, welches einen darüber informiert, dass man gewonnen hat, und den String `nachricht` (vgl. Code 3 Zeile 5) anzeigt. Die beiden Welten haben unterschiedliche Sieggegenstände, sodass es auf dem Kiez erforderlich ist, das eigene Portmonnaie wiederzufinden.
Weitere Möglichkeiten, das Spiel zu beenden, sind, deine Lebenspunkte auf 0 sinken, oder eine festgelegte Zeit - bisher 5 Minuten - ablaufen zu lassen.

2.5 Kommandos

Im Allgemeinen wurde jedem Gegenstand mindestens ein X-Kommando[4] zugeordnet. Diese Kommandos werden alle beim Einlesen aus den XML-Dateien in Listen gespeichert und dann den Gegenständen hinzugefügt. Jeder Gegenstand hält damit seine eigene Liste an Kommandos, die abgefragt werden können. So wird sichergestellt, dass eine Aktion nur ausgeführt wird, wenn der entsprechende Gegenstand auch benutzt wurde.
Die Gegenstände sollten im Laufe des Praktikums immer mehr verschiedene Funktionen erhalten. Dies war insofern schwierig, als dass die zu implementierenden Funktionen immer komplexer wurden[5] und häufig auch ein Gegenstand eine ganz andere Sache können sollte, als bisher, ohne jedoch die alte(n) Funktion(en) einzubüßen.
Dafür wurde sich des Kommando-Entwurfsmusters bedient. Dies funktioniert wie folgt:

[1] s. Kapitel 2.5
[2] *eXtensible Markup Language*, findet Verwendung in der Strukturierung von Daten
[3] *HyperText Markup Language*, findet Gebrauch bei der Gestaltung von Web-Oberflächen
[4] z.B. Lebenscommand zur Erhöhung der Energie, oder Beendencommand zum Beenden des Spiels
[5] So gibt es z.B. eine Zeitmaschine, mit der man exakt 5 Räume zurückreisen kann.

Es gibt eine Befehlsklasse (Command) und eine Menge an konkreten Befehlen (Lebenscommand, Beendencommand, etc.). Ein Klient erzeugt eine Aufruf an der Befehlsklasse und diese leitet diesen Aufruf dann an an einen oder mehrere konkrete Befehle weiter. Diese konkreten Befehle führen dann die entsprechenden Schritte aus.

```
1 public interface Command {
2     public void apply();
3 }
```

Code 4: Unser Command-Interface

Dieses Entwurfsmuster bietet den Vorteil, dass wir die Wirkung der Gegenstände in XML-Dateien auslagern können, weil das Programm intern ein Interface benutzt und wir so die Implementation erst später angeben dürfen.
So sieht z.B. das Lebenscommand, welches natürlich das Interface Command implementiert, so aus:

```
1  public LebenCommand(Spieler spieler, int hp)
2      {
3          _hp = hp;
4          _spieler = spieler;
5      }
6
7      @Override
8      public void apply()
9      {
10         _spieler.getLebenspunkte()
11             .veraendereLebenspunkte(_hp);
12     }
```

Code 5: LebenCommand - eine implementation des Command-Interfaces

In diesem wird die `apply`-Methode überschrieben und verändert jetzt das Leben des Spielers. So kann jedes Mal, wenn mit einem Gegenstand interagiert wird, die Methode `benutzeGegenstand` aus der Klasse `Gegenstand` aufgerufen werden.

```
1 public void benutzeGegenstand() {
2     if (!_commandList.isEmpty()) {
3         ArrayList<Command> iterierListe = _commandList;
4         for (Command command : iterierListe) {
5             command.apply();
6         }
7     }
8 }
```

Code 6: die benutzeGegenstand-Methode aus der Klasse Gegenstand

Sie geht alle Commands durch, die in dem Gegenstand gespeichert sind, und führt

sie nacheinander aus[1], ohne dass ihre konkrete Implementation - also die Funktion, die der Gegenstand hat - von Bedeutung für die Funktionalität dieser Methode ist. Somit lässt sich ganz einfach eine neue Funktion zu einem Gegenstand hinzufügen, indem einfach bei der Erstellung ein weiteres Command der Liste hinzugefügt wird, welches nun beim Benutzen des Gegenstandes abgearbeitet wird.

3 Fazit

Es wurde kurz dargestellt, wobei es sich bei *KUUL* handelt, worauf es aufbaut (*Zuul*), und es wurden die im Spiel vorhandenen Gegenstände behandelt. Im Bezug auf die Gegenstände wurde das Lagerungssystem *Rucksack* angesprochen, die Funktionalität des Vergammels wurde dargelegt, die Umsetzung der Energieregeneration wurde gezeigt, und die Kombination von Gegenständen wurde besprochen. Des Weiteren wurde auf das Kommando-Entwurfsmuster eingegangen und die spezifische Anwendung innerhalb des *KUUL*-Systems dargelegt. Auch die Speicherung und Erstellung der Gegenstände mithilfe von XML wurde angesprochen.
Die XML-Speicherung ermöglicht ohne großen Arbeits- und Zeitaufwand die Erstellung getrennter Gegenstandssammlungen für unterschiedliche Welten sowie die beliebige Erweiterung vorhandener Sammlungen.
Das im Praktikum entstandene Spiel *KUUL* ist somit beliebig erweiterbar.

[1] Die `iterierListe` ist nur dazu da, um Fehlern vorzubeugen, die entstehen würden, wenn ein `Command` in seiner `apply`-Methode die Liste verändert. So z.B. die vergammelbaren Gegenstände.