Design Dokument
Übungsbeispiel 2

Softwareentwicklungspraktikum
SS 2011 Gruppe 6383

Gigler Peter #1030125 Hörandner Felix #1031040
Kühnel Johannes #1030123 Prokop Lukas #1031367

31. März 2011

1 Aufgabenstellung

Die Aufgabenstellung dieser Übung zielt auf das Erlernen des Umgangs mit Datenbanken in Form von Binärdateien ab und versucht diese Aufgabe mit einem objektorientierten Ansatz darzulegen.

2 Der Ablauf in der main.cpp

Die main-Methode ist dafür zuständig den Standardprompt auszugeben. Bei Benutzereingaben sucht sie den zuständigen Befehl und gibt an ihn die Kontrolle ab. Um zu erkennen welcher der richtige Befehl ist wird die `isResponsible`-Methode aufgerufen. Bei Erfolg wird anschließend die `execute`-Methode aufgerufen und die Kontrolle an den Befehl abgegeben. Dieser kann nun weitere Parameter einlesen und Operationen zB. an der Datenbank durchführen. Falls kein Befehl gefunden wurde, wird die entsprechende Fehlermeldung ausgegeben.

3 Der Dex Prompt

Nach dem Start des Programms wird der Dex Prompt ausgeworfen. Dieser erwartet Kommandos, um das Programm zu bearbeiten.

```
dex >
```

Die akzeptieren Befehle sind in der Aufgabenstellung spezifiziert[1].

[1] `http://media.iicm.tugraz.at/wiki/SEP/EX2_SS11`

4 Makefile

Das Makefile wurde so gebaut, dass es grundsätzlich vier verschiedene Ziele erkennt:

- `run` baue und starte die executable Datei
- `tar` erzeuge ein tar-Archiv mit allen Quelldateien
- `valgrind` teste die Implementierung mit valgrind
- `clean` räume erzeugte Dateien auf

5 Klassenhierarchie

5.1 Klassen

5.1.1 `UserInterface`

Die UserInterface stellt dem Programmierer Basiswerkzeuge zur Benutzer-Computerschnittstelle zur Verfügung. Der Konstruktor benötigt keine Argumente. Danach sind die Methoden `readString` und `stringToUnsignedInt` verfügbar, die einen String vom cin lesen und einen String zu einer Ganzzahl ohne Vorzeichen konvertieren. Weitere Methoden sind `readYN` und `readUnsignedInt`, die eine Ja/Nein Frage bis zu einer validen Antwort stellen und eine Ganzzahl ohne Vorzeichen einliest.

5.1.2 `Database`

Die Datenbankklasse ist für die Interaktion mit den binären Textdateien zuständig. Sie hat hierfür mehrere getter und setter Methoden, welche mit Attacken- und Monsterobjekten arbeiten.

`read` Die Methode `read` nimmt einen Dateinamen (`string`) als Dateiname und versucht aus diesem entsprechend dem Dex Format Daten einzulesen.

`write` `write` bildet das Gegenstück zu `read` und schreibt die existierende (aktuell geladene) Datenbank in eine Datei (Dateiname als `string` bildet Parameter).

`sync` Diese Methode wird verwendet, um die versch. Dexe miteinander zu synchronisieren. Diese Methode übernimmt Aufgaben des Zusammenfügens der Dexe.

`toggle` Diese Bonus-Methode erlaubt es zwischen den beiden Dexen zu wechseln.

`add*` Die Klasse stellt Methoden zur Verfügung, um Monster und Attacken zur Datenbank hinzuzufügen.

`get*` getter Methoden stehen bereit, um auf den gesamten Satz von Monstern und Attacken zuzugreifen. Der primäre Dateiname kann über die Methode `getPrimaryFile` abgerufen werden.

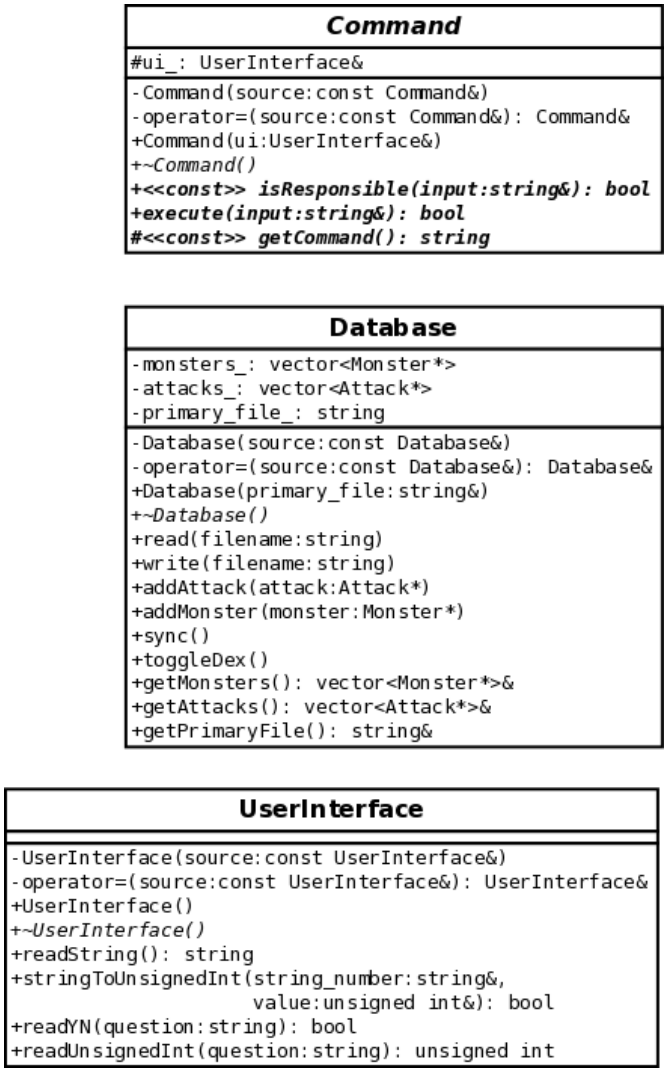

Abbildung 1: Die Basisklassen Teil 1

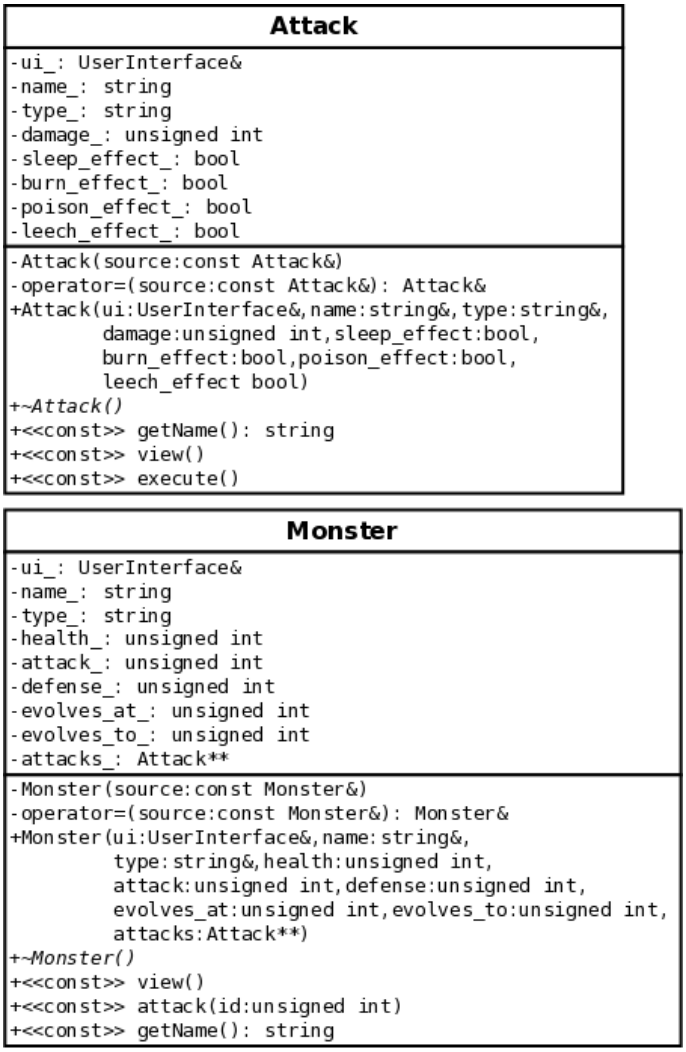

Abbildung 2: Die Basisklassen Teil 2

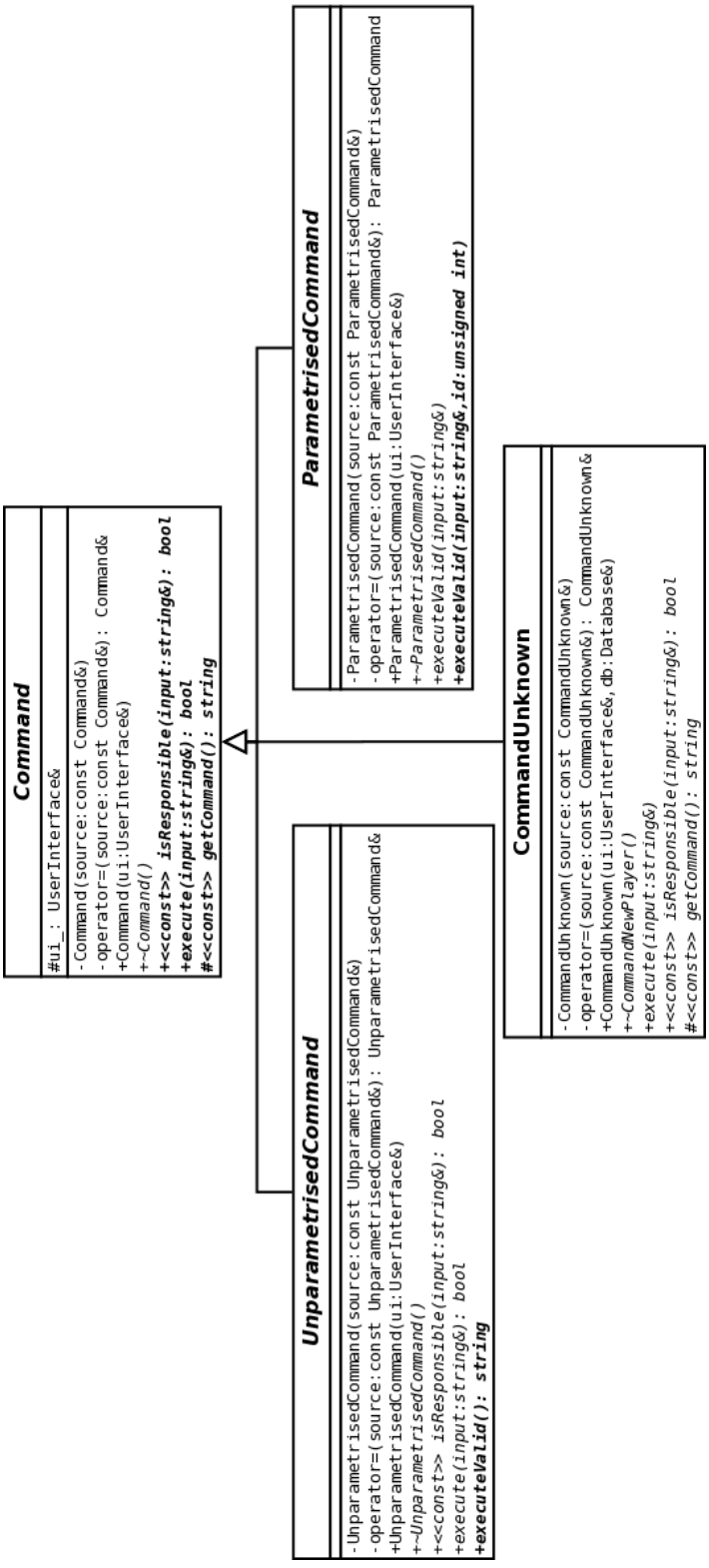

Abbildung 3: Die Command Basisklassen

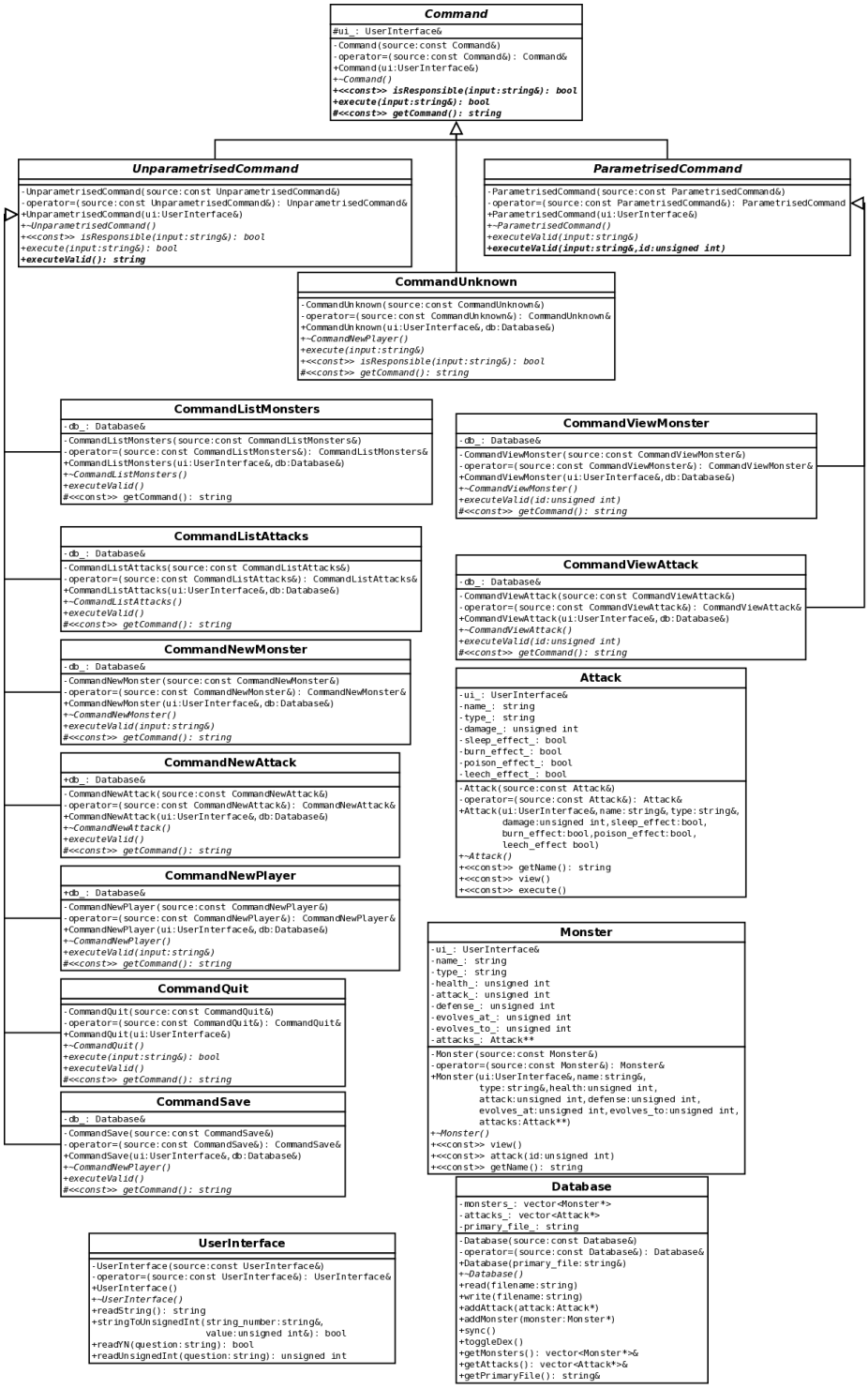

Abbildung 4: Die gesamte Klassenhierarchie

5.1.3 Attack

Wie der Name schon ausdrückt, steht die `Attack` Klasse für eine Attacke. Sie bildet eine Basisklasse, welche mit speziellen Attributen wie `name_` und `burn_effect_` ausgestattet ist.

`getName` Getter Methode für das `name_` Attribut

`view` Gibt Statusinformationen über die Attacke auf der Konsole aus

`execute` Führt die spezifizierte Attacke aus.

5.1.4 Monster

Die Monsterklasse nimmt im Konstruktor die Parameter `ui` (Referenz auf ein `UserInterface` Objekt), `name`, `type` (Typ des Monsters), `health` (Gesundheitspunkte), `attack` (Attackenstärke), `defense` (Verteidigungskraft), `evolves_*` (zu welchem Monster es evolutioniert bzw. evolutioniert ist) und `attacks`, welches einen Doppelpointer auf Attacken darstellt, welches das Monster ausführen kann.

Diese Klasse besitzt wie `Attack` eine Methode `view` zum Anzeigen der Statusinformation und `attack` zum Exekutieren einer Attacke (ID der Attacke bildet Parameter). `getName` ist eine getter Methode für die `name_` Eigenschaft.

5.1.5 Die Command Klassen

Bei der `Command` handelt es sich um eine abstrakte Klasse. Das Interface wird von `UnparametrisedCommand` und `ParametrisedCommand` implementiert, die Subklassen sind. Es verlangt die Methode `isResponsible`, die überprüft, ob die vorliegende Klasse für das gegebene Kommando zuständig ist. `execute` ist dafür verantwortlich, dass die entsprechenden Objektmethoden aufgerufen werden und der Befehl abgearbeitet wird. Die Methode `getCommand` retourniert nur den Prompt-Befehl als String. Dieses Interface wird von den folgenden Klassen (sie entsprechen den Prompt-Befehlen) implementiert:

- `CommandList`
- `CommandListAttacks`
- `CommandListMonsters`
- `CommandNewAttack`
- `CommandNewMonster`
- `CommandNewPlayer`
- `CommandQuit`
- `CommandSave`
- `CommandUnknown`
- `CommandViewAttack`

- `CommandViewMonster`
- `ParametrisedCommand`
- `UnparametrisedCommand`

6 Nachwort

Die Graphiken wurden durch LaTeX leider in miserabler Qualität eingebunden. Eine Onlineversion der Dia-Quelldatei befindet sich auf sep_ex2.dia.