



Hausaufgabe 6 – Produkttypen, algebraische Datentypen und interaktive Programme

Das Entwurfsrezept hilft uns Programme zu schreiben, indem wir der Struktur unserer Daten folgen. Dafür ist es wichtig, geeignete Datendefinitionen zu wählen. Auch beim Entwurf von Datentypen kann man bottom-up oder top-down vorgehen. Die Vor- und Nachteile sind ganz ähnlich wie bei dem Entwurf von Programmen.

In dieser Hausaufgabe üben Sie Summen- und Produkttypen zu kombinieren, um Datendefinitionen für Anwendungen zu entwerfen. Außerdem nutzen wir Produkttypen um komplexere interaktive Programme zu schreiben.

Für die Bearbeitung dieser Hausaufgabe ist es wichtig, dass Sie die Datei `06-vorlage.rkt` aus dem Forum herunterladen.

Viel Erfolg!

Lernziele

Durch diese Hausaufgabe lernen Sie

- ✓ interaktive Programme mit big-bang zu programmieren,
- ✓ als Zustand der interaktiven Programme Produkttypen zu verwenden,
- ✓ und mit geschachtelten Produkt- und Summentypen zu programmieren.

Nach dieser Übung sollen Sie in der Lage sein, interaktive Programme mit zusammengesetzten Zuständen zu schreiben und selbstständig Datenmodellierungen für Programme mit Produkt- und Summentypen durchzuführen.

Aufgabe 1: Programmieren mit big-bang und Produkten

Bewertungsschema (Gesamtpunktzahl 50 pt)

- ☐ 8pt Es wurde jeweils ein Beispiel für `Speed` (2pt), `Location` (2pt), `WorldState` (2pt) und `MouseEvent` (2pt) angegeben.
- ☐ 8pt `INITIAL-STATE` wurde definiert (2pt) und es wurde jeweils ein Funktionskopf für `draw` (2pt), `tick` (2pt) und `mouse` (2pt) angegeben.
- ☐ 8pt Neben den vorgebenen Tests wurden noch weitere Tests hinzugefügt um eine gute Abdeckung zu erreichen.
- ☐ 8pt Die Funktionen von der Wunschliste sind syntaktisch korrekt, nutzen ihre Eingaben und ihre Ausgaben entsprechend ihrer jeweiligen Signatur.
- ☐ 10pt Das Programm funktioniert korrekt: Eine Katze ist zu sehen und sie verfolgt ihr Futter (5pt). Außerdem wird sie langsamer wenn man auf sie klickt und schneller wenn man auf den Hintergrund klickt (5pt).
- ☐ 8pt Die Funktionen verwenden verständliche Namen und Argumentnamen, geeignete Templates, sind gut lesbar strukturiert und aufgeschrieben. Geeignete Hilfsfunktionen sind definiert. Auch die Hilfsfunktionen haben Signaturen, Aufgabenbeschreibungen und Tests.

In dieser Aufgabe wollen wir interaktiv eine Katze ärgern. Wir werden eine Katze und ihr Futter anzeigen. Das Futter befindet sich an der Position des Mauszeigers. Die Katze verfolgt das Futter mit einer bestimmten Geschwindigkeit.

Schritt 1 des Entwurfsrezepts

Schritt 1 des Entwurfsrezeptes folgend geben wir bereits einige Struktur- und Datendefinitionen vor. Diese finden Sie in der Datei `06-vorlage.rkt`.

Schließen Sie Schritt 1 des Entwurfsrezeptes ab indem Sie noch Beispiele für `Speed`, `Location`, `WorldState` und `MouseEvent` angeben.

Schritt 2 des Entwurfsrezepts

Auch Schritt 2 geben wir bereits teilweise mit folgender Wunschliste vor:

INITIAL-STATE

```
; WorldState  
; An initial state we can start the game with
```

draw

```
; WorldState -> Image  
; Draw the cat and its food at their location
```

tick

```
; WorldState -> WorldState  
; Move the cat towards its food
```

mouse

```
; WorldState Number Number MouseEvent -> WorldState  
; Update the position of the food when the mouse moves
```

Um Schritt 2 abzuschließen definieren Sie `INITIAL-STATE` und geben Sie noch jeweils einen Funktionskopf für `draw`, `tick` und `mouse` an. Wir geben auch schon eine Funktion `cat-game` vor mit der man das Programm starten kann:

```
; WorldState -> WorldState  
; Starts the game  
(define (cat-game initial-state)  
  (big-bang initial-state  
    [to-draw draw]  
    [on-tick tick]  
    [on-mouse mouse]))
```

Probieren Sie ihr Programm im Interaktionsbereich mit `(cat-game INITIAL-STATE)` aus.

Hinweis: Es ist wichtig, dass Ihr Program bei Abgabe im Definitionsbereich keine Aufrufe von `cat-game` enthält.

Schritt 3 des Entwurfsrezepts

Als Anregung für Schritt 3 geben wir schon einmal zwei Tests vor:

```
(check-expect (tick (make-world-state 1 (make-posn 0 0) (make-posn 100 0)))  
              (make-world-state 1 (make-posn 1 0) (make-posn 100 0)))  
(check-expect (mouse (make-world-state 1 (make-posn 0 0) (make-posn 100 0)) 13 14 "move")  
              (make-world-state 1 (make-posn 0 0) (make-posn 13 14)))
```

Schreiben Sie mehr Tests um Schritt 3 abzuschließen.

Schritt 4 des Entwurfsrezepts

Wenn Sie nun in Schritt 4 Templates für die Funktionen anlegen denken Sie daran dass `WorldState` ein Produkttyp und `MouseEvent` ein Summentyp ist.

Schritte 5, 6 und 7 des Entwurfsrezepts

Führen Sie selbstständig Schritte 5, 6 und 7 durch. In Schritt 5 implementieren Sie jeweils den Funktionsbody, in Schritt 6 lassen Sie ihre Tests laufen und in Schritt 7 erhöhen Sie die Qualität Ihres Programms durch Refactoring. Fügen Sie, wenn erforderlich, noch weitere Funktionen zu Ihrer Wunschliste hinzu,

Zusatzaufgabe: Katze beruhigen und anlocken

Erst wenn alles bisher geklappt hat, gehen Sie zu dieser nächsten Phase über - dieser Teil gibt nur 5 von 50 Punkten.

Jetzt wollen wir das Spiel ein wenig interessanter gestalten: Wenn man auf die Katze klickt wird die Bewegung der Katze langsamer (sie wird beruhigt). Wenn man auf den Hintergrund klickt wird die Bewegung der Katze schneller (sie wird angelockt). Um festzustellen ob ein Klick auf der Katze oder auf dem Hintergrund gelandet ist nehmen Sie die Katze als kreisförmig mit einem geeigneten Radius an. Ändern Sie ihr Programm entsprechend. Dazu passen Sie zunächst die Aufgabenbeschreibungen an, fügen dann Tests hinzu die das neue Verhalten abdecken und fehlschlagen und erst danach ändern Sie Ihre Funktionen entsprechend.

Aufgabe 2: Programmieren mit Algebraischen Datentypen

Sie arbeiten für die Spieleschmiede "Blazzird" und sollen nun für das neue Onlinerollenspiel "Lucifer" einen Datenentwurf anfertigen. Von den Game-Designern erhalten Sie folgende Beschreibung:

Ein Spieler kann aus 3 verschiedenen Charakteren wählen: Einen Krieger, eine Zauberin und eine Bogenschützin. Jede dieser Charakterklassen hat ein Primärattribut, was sich u.a. darauf auswirkt, wie stark der verursachte Schaden ist.

Jeder Spieler kann eine Waffe tragen. Jede Waffe hat eine Angabe für einen minimalen Schaden (MinDmg), einen maximalen Schaden (MaxDmg) und eine Angriffsfrequenz (APS -- Attacks per Second).

Neben der Waffe, die ein Spieler tragen kann, hat jeder Spieler auch einen Level (eine Zahl, mindestens 1 und maximal 60) und folgende Attribute:

- Stärke (Primärattribut für Krieger)
- Geschicklichkeit (Primärattribut für Bogenschützin)
- Intelligenz (Primärattribut für Zauberin)
- Vitalität
- Energie

Diese sind jeweils ganze Zahlenwerte zwischen 0 und 200.

In dieser Aufgabe werden Sie bereits schon einmal die ersten drei Funktionen `damage-per-second`, `max-mana` und `max-life` für das Spiel programmieren.

Kollegen haben bereits folgende Datendefinitionen (teilweise mit noch auszufüllenden Löchern) programmiert.

```
; A Player is ...
; ...
; interp. the player of a role playing game.
(define WARRIOR-WITH-WEAPON ...)
(define AMAZON-WITH-WEAPON ...)
(define MAGE-WITH-WEAPON ...)
(define WARRIOR-WITHOUT-WEAPON ...)
(define AMAZON-WITHOUT-WEAPON ...)
(define MAGE-WITHOUT-WEAPON ...)

; An Attribute is a Number that falls into the interval:
; - greater or equal than 0 and less or equal than 200
; interp. attributes of a player that change the abilities of its character.

; DPS is a Number greater than 0.
; interp. the damage a player can inflict per second.
(define DPS-WITHOUT-WEAPON 0)
```

1. Datenentwurf mit Produkt- und Summentypen

Dem ersten Schritt des Entwurfsrezept folgend, entwerfen Sie zunächst die für die Funktionen relevanten Datentypen.

Füllen Sie hierzu die Lächer in den obigen Datendefinitionen aus und fügen Sie nötige, weitere Datendefinitionen hinzu. Fügen Sie auch noch mehr Beispieldaten zu den existierenden Datendefinitionen hinzu.

Achten Sie dabei auf:

- sinnvolle Interpretationen zu verwenden,
- Beispiele für Ihre Datendefinitionen als Konstanten anzugeben,
- im Falle von Summen und Produkttypen Templates zu schreiben.

Fragen Sie sich bei der Bearbeitung: "Habe ich die obigen Informationen aus dem Text präzise durch Datendefinitionen dargestellt?"

2. Programmieren mit Produkt- und Summentypen

Programmieren Sie nun die folgenden Funktionen auf Basis ihres Datenentwurfs. Gehen Sie dabei jeweils nach dem Entwurfsrezept vor. Das heißt insbesondere:

- folgen Sie allen Schritten des Entwurfsrezepts
 - schreiben Sie Tests
 - verwenden Sie Templates um die Funktionen zu programmieren
 - lassen Sie Ihre Tests laufen
 - wenden Sie am Ende Refactorings an, um evtl. Hilfsfunktionen zu programmieren und ihr Programm zu vereinfachen
- a. Zur Anzeigen in den Spieler-Statistiken, sollen Sie nun eine Funktion schreiben, welche für einen gegebenen Spieler den durchschnittlichen Schaden pro Sekunde (Damage per Second -- DPS) nach folgender Formel berechnet:

$$\text{DPS} = \left(\frac{(\text{MinDmg} + \text{MaxDmg})}{2} \right) \cdot \text{APS} \cdot \left(1 + \frac{\text{Primary}}{100} \right)$$

Ohne Waffe macht ein Spieler keinen Schaden. Kollegen haben bereits folgendes programmiert:

```
; Player -> DPS
; returns the average damage per second a player can perform
(define (damage-per-second player)
  ...)
```

Bewertungsschema (Gesamtpunktzahl 25 pt)

- ☐ 5pt Es wurden Datendefinitionen angegeben, diese sind syntaktisch korrekt und passen zur Aufgabe.
- ☐ 5pt Für Produkttypen wurden entsprechende Strukturdefinitionen angegeben. Diese sind sinnvoll und passen zur Aufgabe.
- ☐ 5pt Für die Datendefinitionen wurden sinnvolle Beispiele als Konstanten definiert.
- ☐ 5pt Für die Datendefinitionen wurden korrekte Template-Funktionen angegeben.
- ☐ 5pt Der Gesamtentwurf deckt die Beschreibung ab und ist in sich konsistent.

Bewertungsschema (Gesamtpunktzahl 25 pt)

Die drei Funktionen wurden implementiert und:

- ☐ 10pt es wurden Tests angegeben, die Tests sind sinnvoll, spiegeln das erwartete Verhalten wider und die Testabdeckung ist gut.
- ☐ 5pt das Programm ist syntaktisch korrekt, verwendet die richtigen Typen und Tests laufen erfolgreich durch.
- ☐ 10pt Schritt 7 des Entwurfsrezepts wurde erfolgreich durchgeführt, sinnvolle Konstanten und Hilfsfunktionen sind definiert, getestet und kommentiert. Vereinfachungen wurden erkannt und durchgeführt.

- b. Für jeden Spieler kann auch das maximale Mana und die maximalen Lebenspunkte, die man mit dem aktuellen Level haben kann, berechnet werden. Da sich diese für die verschiedenen Klassen unterscheiden (und z.B. eine Zauberin mehr Mana haben kann als ein Krieger), werden Klassenboni angewandt. Auch die Ausgangswerte (bei Spielanfang) unterscheiden sich von Klasse zu Klasse. Diese können aus der folgenden Tabelle entnommen werden:

Klasse	Energie		Lebenspunkte	
	Start-Energy	Energy-Class-Bonus	Start-Vit	Vit-Class-Bonus
Krieger	10	1	40	4
Zauberin	35	2	15	2
Bogenschützlin	15	1.5	20	3

Schreiben Sie eine Funktion, welche die maximalen Energiepunkte nach folgender Formel berechnet:

$$\text{MaxMana} = (\text{Energy} - \text{Start-Energy} + \text{Level} - 1) \cdot \text{Energy-Class-Bonus} + \text{Start-Energy}$$

Hierfür wurde bereits programmiert:

```
; Player -> Attribute
; returns the maximum mana of a player
(define (max-mana player)
  ...)
```

- c. Schreiben Sie eine Funktion, welche die maximalen Lebenspunkte nach folgender Formel berechnet:

$$\text{MaxLife} = (\text{Vitality} - \text{Start-Vit} + \text{Level} - 1) \cdot \text{Vit-Class-Bonus} + \text{Start-Vit}$$

Auch hierfür wurde bereits programmiert:

```
; Player -> Attribute
; returns the maximum life of a player
(define (max-life player)
  ...)
```