Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Simon Dorer, Sebastian Klähn

Universität Freiburg
Institut für Informatik
Wintersemester 2024

Übungsblatt 6

Abgabe: Montag, 25.11.2024, 9:00 Uhr

Erinnerung: Chat

Neben der Vorlesung und den Übungsblättern steht Ihnen ein separater Chat (hier) zur Verfügung. Dort können Sie jederzeit Fragen zu den Vorlesungsinhalten, Übungsblättern oder anderen Themen rund um die Veranstaltung stellen. Der Chat bietet eine einfache Möglichkeit, Unklarheiten schnell zu klären und ermöglicht es Ihnen, von den Fragen und Antworten Ihrer Kommilitonen zu profitieren.

Hinweis: Funktionen und Datenklassen

In diesem Übungsblatt sollen Sie Funktionen erstellen, die mit Instanzen von Datenklassen arbeiten. Diese Funktionen sollen **neue Instanzen** der Datenklasse zurückgeben, ohne die übergebenen Argumente zu verändern.

Angenommen, wir haben die folgende Datenklasse `Foo` definiert:

```
@dataclass
class Foo:
    x: int
    t: list[str]
```

Nun definieren wir eine Funktion `bar`, die eine Instanz von `Foo` entgegennimmt und eine **neue** Instanz von `Foo` erstellt:

```
def bar(f: Foo) -> Foo:
    return Foo(f.x + 1, f.t + ["extra"])
```

Wenn wir diese Funktion mit einer Instanz von `Foo` aufrufen, z. B.:

```
f = Foo(1, ["Hallo", "Welt"])
```

dann bleibt die ursprüngliche Instanz `f` unverändert. Nach dem Funktionsaufruf muss gelten:

```
f.x == 1
f.t == ["Hallo", "Welt"]
```

Das bedeutet, dass Ihre Funktionen die Eingabe **nicht direkt verändern** dürfen. Stattdessen sollen sie auf der Grundlage der Eingabe neue Instanzen der Datenklasse erzeugen. Dies gilt für alle Aufgaben in diesem Übungsblatt!

Aufgabe 6.1 (Snake; 20 Punkte; Datei: `snake.py`)

In dieser Aufgabe programmieren Sie eine Version des Spiels Snake. In Ihrem Repository[1] (Gitea) befinden sich in dem Ordner zu Blatt 06 bereits zwei Dateien. In `game_snake.py` befindet sich ein Grundgerüst des Spiels, welches von Ihnen weder verändert noch verstanden werden muss. Sie benötigen diese Datei lediglich um das Spiel zu starten (indem Sie diese Datei ausführen). In `test_snake.py` stellen wir Ihnen Tests bereit, mit denen Sie ihre einzelnen Funktionen und Datenklassen überprüfen können (indem Sie die Datei ausführen).

Erstellen Sie eine neue Datei `snake.py` in der Sie die folgenden Aufgaben bearbeiten.

(a) **Datenklasse Vec2D; 1.5 Punkte**

Schreiben Sie eine Datenklasse `Vec2D`, die ein Attribut `x` für die x-Koordinate und ein Attribut `y` für die y-Koordinate eines 2D-Vektors besitzt. Beide Koordinaten sollen ganzzahlig sein.

```
>>> v = Vec2D(52, 20)
>>> v.x
52
>>> v.y
20
```

(b) **Vektoren addieren; 1 Punkt**

Schreiben Sie eine Funktion `add_vecs`, die zwei Vektoren als Argumente nimmt, diese komponentenweise addiert und das Ergebnis als neuen Vektor zurückgibt.

```
>>> v1 = Vec2D(43, 10)
>>> v2 = Vec2D(-4, 20)
>>> add_vecs(v1, v2)
Vec2D(x=39, y=30)
```

(c) **Datenklasse Item; 1.5 Punkte**

Schreiben Sie eine Datenklasse `Item` mit den Attributen `position` und `energy`. Dabei soll `position` ein Vektor sein, der die Position des Items beschreibt und `energy` eine ganze Zahl, die beschreibt, um wie viel eine Schlange wächst wenn sie dieses Item aufsammelt.

```
>>> item = Item(Vec2D(4, 2), 3)
>>> item.position
Vec2D(x=4, y=2)
>>> item.energy
3
```

[1] Verwenden Sie `git pull` um Ihr lokales Repository zu aktualisieren oder laden Sie die Dateien direkt über Gitea (https://git.laurel.informatik.uni-freiburg.de/2024WS-EidP/) herunter.

(d) **Datenklasse Snake; 1.5 Punkte**

Schreiben Sie eine Datenklasse `Snake` mit den folgenden Attributen: `positions`, `direction`, `alive` und `grow`. `positions` ist eine Liste von Vektoren, die beschreiben, an welchen Positionen sich Segmente der Schlange befinden. `direction` ist ein Vektor, der beschreibt in welche Richtung sich die Schlange bewegt. `alive` ist ein Wahrheitswert, der beschreibt ob die Schlange lebt oder nicht, und `grow` ist eine ganze Zahl, die beschreibt, um wie viele Segmente die Schlange noch wächst.

```
>>> s = Snake([Vec2D(0, 0), Vec2D(1, 0)], Vec2D(-1, 0), True, 6)
>>> s.positions
[Vec2D(x=0, y=0), Vec2D(x=1, y=0)]
>>> s.direction
Vec2D(x=-1, y=0)
```

Und so weiter...

(e) **Datenklasse Game; 1.5 Punkte**

Schreiben Sie eine Datenklasse `Game` mit den folgenden Attributen: `snake`, `width`, `height`, `frame` und `items`. `snake` ist eine Schlange. `width` und `height` sind ganze Zahlen, die die Spielfeldgröße bestimmen. `frame` ist eine ganze Zahl, die die bisherigen Frames zählt und `items` ist eine Liste von Items, die sich auf dem Spielfeld befinden.

```
>>> g = Game(Snake([Vec2D(3, 0), Vec2D(2, 0)], Vec2D(1, 0), True, 2),
...                 16, 14, 0, [Item(Vec2D(7, 2), 3)])
>>> g.height
14
>>> g.width
16
```

Und so weiter...

(f) **Schlange drehen; 3 Punkte**

Schreiben Sie eine Funktion `turn_direction` die einen Vektor `direction` und eine ganze Zahl `turn` als Argumente nimmt und einen neuen Vektor zurück gibt. Der Vektor `direction` zeigt in eine von vier möglichen Richtungen: Nach Rechts (`Vec2D(1, 0)`), nach Unten (`Vec2D(0, 1)`), nach Links (`Vec2D(-1, 0)`) oder nach Oben (`Vec2D(0, -1)`)[2]. Ist `turn 1`, so soll der um eine Richtung im Uhrzeigersinn gedrehte Vektor zurückgegeben werden. Ist `turn -1`, so soll der gegen den Uhrzeigersinn gedrehte Vektor zurückgegeben werden. Ist `turn` eine andere Zahl, so soll `direction` unverändert zurückgegeben werden.

Im Uhrzeigersinn: oben→ rechts→ unten→ links→...
Gegen den Uhrzeigersinn: oben→ links→ unten→ rechts→...

[2]Der Ursprung des verwendeten Koordinatensystems wird oben links dargestellt. Das ist eine gängige Konvention.

Schreiben Sie eine Funktion `turn_snake`, die eine Schlange `snake` und eine Zahl `turn` als Argumente nimmt und eine Schlange zurückgibt. Ist die Schlange `snake` tot, so soll die Schlange unverändert zurückgegeben werden. Ist die Schlange lebendig, so soll eine neue Schlange zurückgegeben werden, deren Blickrichtung mithilfe von `turn_direction` gedreht wurde. Alle anderen Attribute der neuen Schlange sollen unverändert von `snake` übernommen werden.

```
>>> position = Vec2D(1, 0)
>>> turn_direction(position, 1)
Vec2D(x=0, y=1)
>>> turn_direction(position, -1)
Vec2D(x=0, y=-1)
>>> turn_direction(position, 7)
Vec2D(x=1, y=0)
```

(g) **Schlange bewegen; 4 Punkte**

Schreiben Sie eine Funktion `grow_positions`, die eine Liste von Vektoren `positions` und einen Vektor `direction` als Argumente nimmt und eine neue Liste von Vektoren zurückgibt. Die neue Liste soll dabei aus einem neuen Vektor, gefolgt von allen Vektoren aus `positions` bestehen. Der neue Vektor entsteht durch Addition des ersten Vektors von `positions` und der `direction`.

Schreiben Sie eine Funktion `move_snake`, die eine Schlange `snake` als Argument nimmt und eine neue Schlange mit geändertern Positionen zurückgibt. Die Funktion soll folgendes Verhalten haben: Wenn die Schlange nicht lebt, ändern sich die Positionen nicht. Lebt die Schlange, so können die *neuen Positionen* durch `grow_positions` berechnet werden (die Schlange soll sich in ihre Blickrichtung bewegen). Ist `grow` von der `snake` gleich 0, so sollen außer der letzten Position, alle *neuen Positionen* zu der neuen Schlange hinzugefügt werden. Ist `grow` von der `snake` hingegen größer als 0, so sollen alle *neuen Positionen* übernommen werden und `grow` der neuen Schlange um eins kleiner sein als das von `snake`.

```
>>> s = Snake([Vec2D(0, 0)], Vec2D(1, 0), True, 1)
>>> s = move_snake(s)
>>> s.positions
[Vec2D(x=1, y=0), Vec2D(x=0, y=0)]
>>> s.grow
0
>>> s = move_snake(s)
>>> s.positions
[Vec2D(x=2, y=0), Vec2D(x=1, y=0)]
>>> s.grow
0
```

(h) **Kollisionen; 2 Punkte**

Schreiben Sie eine Funktion `collision`, die eine Schlange `snake`, die Spielfeld-

breite `width` und die Spielfeldhöhe `height` (beide ganzzahlig) als Argumente nimmt. Die Funktion soll zurückgeben, ob die Schlange kollidiert oder nicht. Eine Kollision liegt in den folgenden beiden Fällen vor:

- Der Kopf (erster Vektor von `positions`) der Schlange hat die gleiche Position wie ein Segment ihres eigenen Körpers (alle bis auf den ersten Vektor von `positions`).
- Der Kopf der Schlange ist auf einer Position außerhalb des Spielfelds.

Andernfalls gibt es keine Kollision.

```
>>> s = Snake([Vec2D(9, 9)], Vec2D(1, 0), True, 0)
>>> collision(s, 10, 10)
False
>>> collision(s, 10, 9)
True
```

(i) **Items generieren; 2 Punkte**

Schreiben Sie eine Funktion `generate_item`, die ein Spiel-Objekt `game` als Argument nimmt und ein Item mit zufälligen Werten zurückgibt. Das Item darf nicht außerhalb des Spielfelds liegen und die Energie muss im Intervall [1, 5] liegen. Verwenden Sie die Funktion `randint` des Moduls `random`, um eine zufällige Zahl im jeweiligen Intervall zu generieren. Die Funktion `randint` nimmt zwei ganze Zahlen `a` und `b` als Argumente und gibt eine zufällige Zahl im Intervall [`a`, `b`] zurück.

(j) **Items aufsammeln; 2 Punkte**

Schreiben Sie eine Funktion `pick_item`, die eine Liste von Items `items` und einen Vektor `position` als Argumente nimmt. Die Funktion soll eine Kopie der Liste mit allen Items erstellen, die sich nicht auf der Position `position` befinden. Zudem soll die Energie aller Items aufsummiert werden, die sich auf `position` befinden. Die Funktion soll die neue Liste und die Energiesumme (ganzzahlig) als Tupel zurückgeben.

Aufgabe 6.2 (Snake spielen)

Diese Aufgabe ist optional! Um die volle Punktzahl zu erreichen, müssen Sie PyGame nicht installieren. In der vorherigen Aufgabe haben Sie die Spiellogik von Snake implementiert. Um Snake mit einer Graphischen Oberflache spielen zu können, haben wir bereits ein Python-Skript `game_snake.py` bereitgestellt, das auf Ihren Implementierungen aus Aufgabe 6.1 aufbaut. Sie müssen dieses Skript weder verändern, noch verstehen :)

Zum Ausführen von `game_snake.py` benötigen Sie das Python-Modul `pygame`. Dieses können Sie mit folgendem Befehl über Ihr Terminal installieren:

- Falls Sie Ubuntu verwenden: `sudo apt-get install python3-pygame`
- Falls Sie MacOS verwenden: `python3 -m pip install -U pygame --user`

- Für alle anderen Betriebssysteme folgen Sie der offiziellen Installationsanleitung: https://www.pygame.org/wiki/GettingStarted#Pygame%20Installation

Um sicherzustellen, dass `pygame` erfolgreich installiert wurde, führen Sie den folgenden Befehl aus:

```
python3.12 -c "import pygame; pygame.__version__"
```

Bei erfolgreicher Installation erhalten Sie eine Ausgabe der folgenden Form[3]:

```
pygame 2.6.1 (SDL 2.28.4, Python 3.12.7)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Nun ist `pygame` erfolgreich installiert. Sie können das Spiel starten, indem Sie die `game_snake.py` Datei ausführen. Mit den Tasten A und D (oder mit der linken und rechten Pfeiltaste) können Sie die Schlange nach links und rechts drehen.

Viel Spaß beim Spielen :D

Aufgabe 6.3 (Erfahrungen; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `7.5 h` steht für 7 Stunden 30 Minuten.

Der Build-Server überprüft ebenfalls, ob Sie das Format korrekt angegeben haben. Prüfen Sie, ob der Build-Server mit Ihrer Abgabe zufrieden ist, so wie es im Video zur Lehrplattform gezeigt wurde.

[3] Falls Sie Probleme bei der Installation haben, können Sie sich gerne im Chat melden :)