Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Lukas Kleinert, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2023

Übungsblatt 5

Abgabe: Montag, 20.11.2022, 9:00 Uhr morgens

Typannotationen

Ab diesem Übungsblatt müssen Sie bei jeder Funktionsdefinition korrekte Typannotationen für die Argumente und den Rückgabewert angeben. Bei Verstößen gibt es bis zu einem Punkt Abzug pro Funktionsdefinition.

Pip und pygame installieren

Für dieses Übungsblatt benötigen Sie das Python-Modul `pygame`.

Um neue Module in Python installieren zu können, benötigen Sie zuerst den packageinstaller `pip`. Diesen werden Sie in den kommenden Wochen auch verwenden um andere Module zu installieren.

- **pip installieren (WSL/Ubuntu)**

  Führen Sie die folgenden Befehle in einem Terminal aus:

  ```
  curl -sSL https://bootstrap.pypa.io/get-pip.py -o get-pip.py
  python3.12 get-pip.py
  rm -f get-pip.py
  python3.12 -m pip install pip setuptools -U
  ```

  Sie sollten nun mit dem Befehl

  ```
  python3.12 -m pip --version
  ```

  eine Ausgabe bekommen, die ungefähr so aussieht:

  ```
  pip 23.3.1 from <...> (python 3.12)
  ```

- **pygame installieren**

  ```
  python3.12 -m pip install pygame
  ```

Aufgabe 5.1 (Python Games; 20 Punkte; Datei: `snake.py`)

In dieser Aufgabe programmieren Sie eine Version des Spiels Snake. In Ihrem Repo[1] (Gitea) befinden sich in dem Ordner zu Blatt 05 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 Vec2; 1.5 Punkte**

Schreiben Sie eine Datenklasse `Vec2`, 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.

(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.

(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.

(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.

(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.

[1] Verwenden Sie `git pull` um Ihr lokales Repo zu aktualisieren oder laden Sie die Dateien direkt über Gitea (https://git.laurel.informatik.uni-freiburg.de/2023WS-EiP) herunter.

(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 (`Vec2(1, 0)`), nach Unten (`Vec2(0, 1)`), nach Links (`Vec2(-1, 0)`) oder nach Oben (`Vec2(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→...

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.

(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`.

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

(h) **Kollisionen; 2 Punkte**

Schreiben Sie eine Funktion `collision`, die eine Schlange `snake`, die Spielfeldbreite `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[3] der Schlange hat die gleiche Position wie ein Segment ihres eigenen Körpers[4].
- Der Kopf der Schlange ist auf einer Position außerhalb des Spielfelds.

Andernfalls gibt es keine Kollision.

(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.

Wenn Sie alle Teilaufgaben korrekt bearbeitet haben, sollten Sie ein spielbares Programm haben. Steuern Sie die Schlange mit den Tasten A/D oder den Pfeiltasten Links/Rechts :)

Aufgabe 5.2 (Erfahrungen; 0 Punkte; 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 Webplatform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `3.5 h` steht dabei für 3 Stunden 30 Minuten.

[3] Der erste Vektor von `positions`.
[4] Alle bis auf den ersten Vektor von `positions`.