Einführung in die Programmierung

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

Universität Freiburg
Institut für Informatik
Wintersemester 2024

Übungsblatt 8

Abgabe: Montag, 09.12.2023, 9:00 Uhr

Aufbau des Übungsblatts

In der Vorlesung haben Sie Methoden und Vererbung von Datenklassen kennengelernt. Seien zum Beispiel zwei Datenklassen `A` und `B` gegeben, und `B` erbt von `A`, dann besitzt `B` mindestens alle Methoden von `A`. Bei der Definition von `B` können Sie Methoden von `A` überschreiben oder diese mit `super` explizit aufrufen. Während wir Ihnen vorgeben, welche Datenklassen und Methoden es geben muss, müssen Sie oft selbst entscheiden, wann Datenklassen erben oder Methoden überschreiben. Versuchen Sie durch Vererbung zu erreichen, dass stets so wenig Code wie möglich dupliziert wird. Sie dürfen *keine* Fallunterscheidungen (z.B. `isinstance` oder `match`) auf dem Parameter `self` verwenden. Überschreiben Sie stattdessen Methoden, um unterschiedliches Verhalten für Subklassen umzusetzen. Fallunterscheidungen auf anderen Parametern sind erlaubt.

Aufgabe 8.1 (Python Games + OOP; 20 Punkte; Datei: `asteroids.py`)

In dieser Aufgabe entwickeln Sie eine Variante des beliebten Atari Spiels *Asteroids*. In dem Spiel steuern Sie ein Raumschiff und versuchen so lange wie möglich den gefährlichen Asteroiden im Weltall auszuweichen. Dafür können Sie sich sowohl geschickt um die Asteroiden herum bewegen, aber das Raumschiff hat auch eine Pistole um Asteroiden zu zerstören. Um das Spiel noch ein bisschen interessanter zu machen, tauchen in regelmäßigen Abständen Items auf, die von dem Raumschiff eingesammelt werden können und entweder die Lebenspunkte oder die Munition des Raumschiffes erhöhen.

In Ihrem Repository[1] (Gitea) befinden sich in dem Ordner zu Blatt 08 bereits zwei Dateien. In `game_asteroids.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_asteroids.py` stellen wir Ihnen Tests zur Verfügung, mit denen Sie einige Teilaufgaben überprüfen können (indem Sie die Datei ausführen). Falls Sie weiterhin Probleme bei der Installation von PyGame haben, können Sie gerne erneut im Chat[2] nachfragen. Alle Teilaufgaben bis auf die letzte sind aber auch ohne PyGame lösbar.

[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.

[2] https://chat.laurel.informatik.uni-freiburg.de/home

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

(a) **Datenklasse Vec2D; 0.5 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 Gleitkommazahlen sein.

(b) **Vektorbetrag; 1 Punkt**

Schreiben Sie eine Methode `abs` der Datenklasse `Vec2D`, die die Länge des Vektors berechnet und als Gleitkommazahl zurückgibt.

(c) **Datenklassen und Vererbung; 5 Punkte**

Schreiben Sie die folgenden Datenklassen, mit den jeweiligen Attributen (Die Attribute sind etwas weiter unten genauer beschrieben). Achten Sie darauf, dass Sie Vererbung sinnvoll verwenden, sodass Attribute nicht mehrfach deklariert werden müssen, wenn diese von mehreren Datenklassen benötigt werden. Dadurch sollte sich eine eindeutige Klassenhierarchie ergeben.

- `GameObject`: `position`, `radius`, `alive`, `color`
- `Projectile`: `position`, `radius`, `alive`, `color`, `speed`
- `StaticObject`: `position`, `radius`, `alive`, `color`, `rotation`
- `Item`: `position`, `radius`, `alive`, `color`, `rotation`, `amount`
- `Ammunition`: `position`, `radius`, `alive`, `color`, `rotation`, `amount`
- `Health`: `position`, `radius`, `alive`, `color`, `rotation`, `amount`
- `Ship`: `position`, `radius`, `alive`, `color`, `shots`, `hp`
- `Asteroid`: `position`, `radius`, `alive`, `color`, `rotation`, `special`

Beachten Sie, dass `Ammunition` und `Health` beide `Items` sind. Die Attribute haben die folgenden Bedeutungen:

- `position`: `Vec2D`, die Position eines Objektes auf dem Bildschirm.
- `radius`: `int`, der Radius (halber Durchmesser) eines Objektes.
- `alive`: `bool`, Objekte werden automatisch aus dem Spiel entfernt, wenn sie nicht mehr leben.
- `color`: `tuple[int,int,int]`, die RGB-Farbe[3] des Objektes.
- `speed`: `float`, Geschwindigkeit mit der sich ein Projektil auf dem Bildschirm fortbewegt.
- `amount`: `int`, Wert eines Items.

[3] https://de.wikipedia.org/wiki/RGB-Farbraum Werte jeweils zwischen 0 und 255

- `shots`: `int`, Anzahl der Munition des Schiffes.
- `hp`: `int`, die Lebenspunkte des Schiffs.
- `rotation`: `float`, Drehwinkel von Objekten in Grad. Kann in Aufgabe f) verwendet werden.
- `special`: `bool`, manche Asteroiden sind besonders.

Beachten Sie auch, dass Sie für die nächsten Aufgaben möglicherweise nicht alle Attribute verwenden müssen.

(d) **Methode update; 4 Punkte**

Fügen Sie den `GameObject`-Datenklassen eine Methode `update` mit den ganzzahligen Parametern `width` und `height` (die Spielfeldgröße), sowie der Gleitkommazahl `delta` ein. Die Methode soll jeweilige Objekte wie folgt verändern: Bei allen `GameObject`-Instanzen soll geprüft werden, ob sich das Objekt innerhalb des Spielfelds[4] befindet, und wenn dies nicht der Fall ist, soll dieses Objekt sterben (das bedeutet `alive = False`). Jedes `Projectile` soll zuerst um `(delta * speed)` nach *oben* bewegt werden.[5] Jedes `StaticObject` soll sich stattdessen zuerst um `delta` nach *unten* bewegen.[6] Zudem soll sich die `rotation` um `(delta / radius)` erhöhen. Ein `Ship` soll prüfen, ob seine Lebenspunkte kleiner oder gleich 0 sind. Ist das der Fall, sollen sie auf `0` gesetzt werden und das Schiff sterben.

(e) **Methode is_colliding; 2 Punkte**

Fügen Sie der Datenklasse `GameObject` eine Methode `is_colliding` hinzu, die neben `self` ein weiteres `GameObject other` als Argument nimmt, und zurückgibt ob die beiden Objekte kollidieren. Die Objekte kollidieren, wenn sich ihre Hitboxen berühren. Die Hitbox eines Objekts ist ein Kreis der durch die Position `position` und den Radius `radius` des Objekts beschrieben wird.

(f) **Methode on_collision; 4 Punkte**

Fügen Sie den `GameObject`-Datenklassen eine Methode `on_collision` hinzu, die ein `GameObject other` als Argument nimmt. Diese Methode bestimmt, was bei einer Kollision mit den Objekten passiert.[7] Die Methode soll folgendes tun: Bei allgemeinen `GameObject`s passiert nichts. Ein `StaticObject` soll bei der Kollision sterben. Ein `Projectile` soll bei der Kollision sterben, wenn das andere Objekt kein `Ship` ist. Ein `Asteroid` soll nicht sterben, wenn das andere Objekt ebenfalls ein `Asteroid` ist. Bei dem Raumschiff gibt es drei Möglichkeiten. Ist das andere Objekt ein `Asteroid`, sollen die Lebenspunkte des Raumschiffes um den `radius` des Asteroids gesenkt werden. Ist das andere Objekt

[4] also `0 <= x < width` und `0 <= y < height`

[5] Erinnern Sie sich daran, dass der Ursprung des Koordinatensystems links oben ist.

[6] Damit es visuell so scheint, als würde sich das Schiff stetig nach vorn bewegen.

[7] Sie müssen diese Methode, sowie `is_colliding`, nicht selbst aufrufen. Der Support-Code ruft die Methode bei einer Kollision zweier Objekte für *beide* Objekte auf.

ein `Health`-Item, sollen die Lebenspunkte um den `amount` des Items steigen. Ist das andere Objekt ein `Ammunition`-Item, steigt `shots` des Raumschiffs um `amount`. Ansonsten passiert nichts.

(g) **Methode shoot; 2 Punkte**

Fügen Sie der Datenklasse `Ship` eine Methode `shoot` hinzu, die ein *rotes* `Projectile` mit Radius `5` und Geschwindigkeit `3.0` zurückgibt. Die Position des `Projectile` ist dabei die aktuelle Position des Raumschiffs. Ist `shots` des Schiffs größer als 0, soll dieser Wert um eins verringert werden. Ist dies jedoch nicht der Fall, soll ein 'leeres' Projektil mit `alive = False` zurückgegeben werden.

(h) **Methode draw; 1.5 Punkte**

Fügen Sie der Datenklasse `GameObject` eine Methode `draw` mit dem Parameter `screen` vom Typ `pygame.Surface` hinzu, die Objekte zeichnet. Verwenden Sie dafür die `pygame`-Funktionen aus dem Modul `pygame.draw`[8]. Zum Beispiel `pygame.draw.circle`. Sie dürfen selbst entscheiden wie das Spiel aussehen soll[9]. Achten Sie jedoch darauf, dass die Größe etwa dem Radius des jeweiligen Objektes entspricht und verschiedene Objekte deutlich voneinander zu unterscheiden sind. Sie dürfen, aber müssen nicht alle Attribute der Klassen verwenden.

Wenn Sie alle Teilaufgaben korrekt bearbeitet haben, sollten Sie ein spielbares Programm haben. Steuern Sie das Schiff mit den Tasten W/A/S/D und schießen Sie mit der Leertaste. Viel Spaß beim Spielen :)

Aufgabe 8.2 (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.

[8] https://www.pygame.org/docs/ref/draw.html
[9] Schönheit wird nicht bewertet