Praktikum (1)

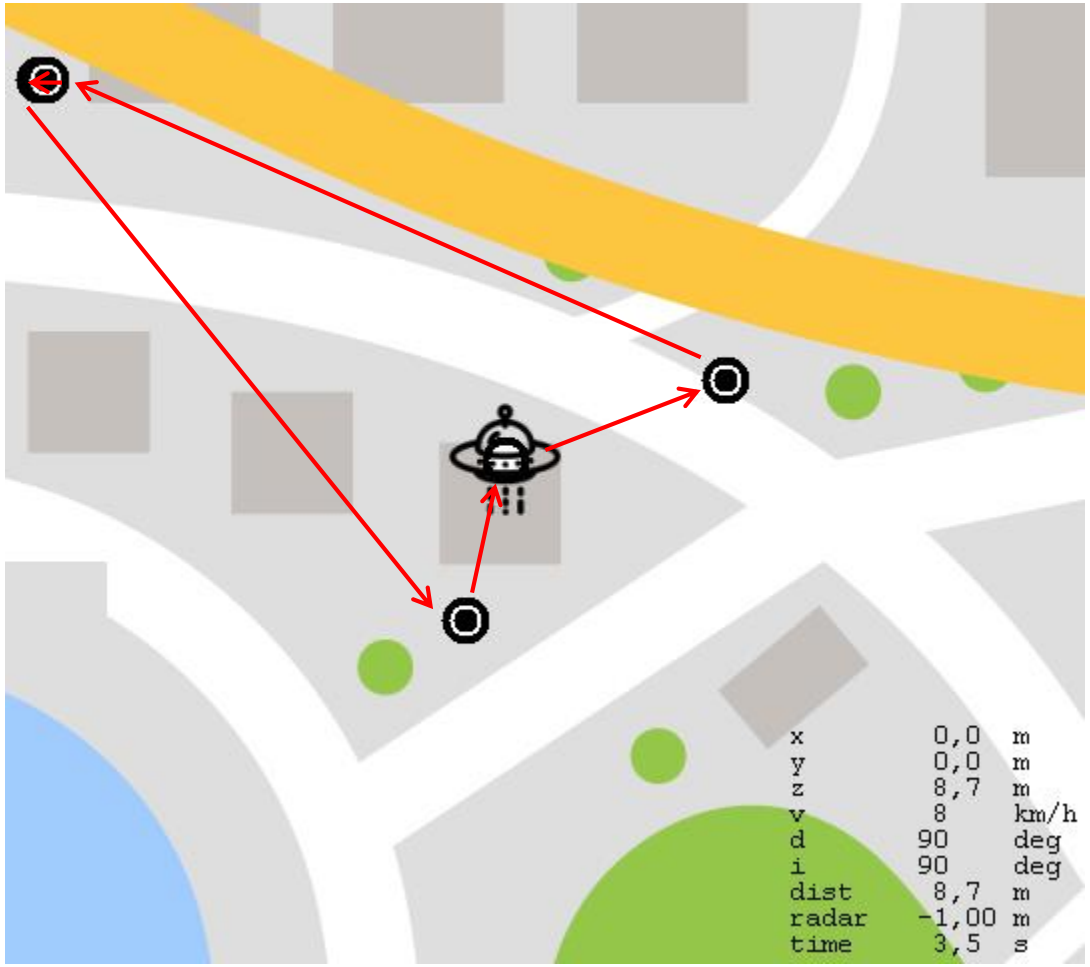
- Im **dritten Teil des Praktikums** sollen folgende Punkte geändert bzw. ergänzt werden:
  - Bisher verwenden wir für Koordinaten (x, y) jeweils zwei float-Werte. Jetzt sollen dafür Tupel genommen werden.
  - Bisher hatten wir ein einziges Ziel, das angeflogen wurde. Jetzt sollen mehrere Ziele angeflogen werden können. Danach soll das Ufo zum Ausgangspunkt wieder zurückkehren. Da es mehrere Ziele gibt, können diese auch in unterschiedlichen Reihenfolgen angeflogen werden.
  - Wir definieren: Eine Route ist eine Liste $[(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$ von Koordinaten. Das Ufo fliegt von (0, 0) nach (x_1, y_1), von (x_1, y_1) nach (x_2, y_2), …, von (x_{n-1}, y_{n-1}) nach (x_n, y_n) und von (x_n, y_n) nach (0, 0). Jedes Flugsegment besteht aus dem Start (takeoff), dem Weiterflug auf der festgelegten Flughöhe (cruising) und den Landen (landing).
  - Durch Umsortierung einer Route erhält man eine weitere Route zwischen den Punkten.

Praktikum (2)

- Wenn wir n Ziele haben, gibt es n! verschiedene Routen. Die Abbildung zeigt eine Beispielroute zum Anfliegen der Ziele (55.0, 20.0), (-116.5, 95.0), (-10.0, -40.0), (-115.0, 95.0). Die Beispielroute ist

  [(55.0, 20.0), (-115.0, 95.0), (-116.5, 95.0), (-10.0, -40.0)]

- Es soll ein Modul ufo_routing.py erstellt werden, das die Routenplanung übernimmt.



Praktikum (3)

Aufgaben

1. Ändern Sie die Funktionen distance(x1, y1, x2, y2), angle(x1, y1, x2, y2), flight_distance(x1, y1, x2, y2, z), fly_to(sim, x, y, z) und cruise(sim, x, y) so, dass die Parameter x1, y1 und x2, y2 und x, y Tupel von zwei Fließkommazahlen sind. z ist die Flughöhe. Die Aufrufe in ufo_main.py und ufo_autopilot.py müssen entsprechend angepasst werden. Die Funktion addDestination verträgt keine Tupel, da sie eine Java-Funktion ist. Deshalb müssen ihr weiterhin zwei Fließkommazahlen übergeben werden.
2. Ersetzen Sie das Ziel (x, y) im Hauptprogramm ufo_main.py durch eine Liste von Zielen. Ein Ziel ist dabei ein Tupel von zwei Fließkommazahlen. Der Ausgangspunkt, der ja auch der Endpunkt ist, soll nicht enthalten sein. Beispiel: [(55.0, 20.0), (-115.0, 95.0), (-116.5, 95.0), (-10.0, -40.0)]
3. In einer Schleife soll sim.addDestination(x, y) mit jedem Punkt der Liste aufgerufen werden.
4. Anstelle des Aufrufs fly_to(sim, x, y, z) im Hauptprogramm ufo_main.py müssen nun alle Ziele in der Liste angeflogen werden. Verwenden Sie eine for-Schleife über die Liste. Danach soll nach (0, 0) zurückgeflogen werden.

5. Zur Bestimmung der zu fliegenden Strecke reicht jetzt nicht mehr ein Aufruf von flight_distance. Implementieren Sie in ufo_autopilot.py die folgende Funktion:

```
def flight_distance_mult(destinations, z):
Parameter:      list destinations : Ziele (Liste von Tupel mit je zwei float)
                float z : Flughöhe
Rückgabewert:   float : Zu fliegende Strecke inklusive Rückweg nach (0, 0)
```

6. Die neue Funktion flight_distance_mult soll nun im Hauptprogramm ufo_main.py anstelle von flight_distance aufgerufen werden.

7. Wenn n Ziele angeflogen werden sollen, ist die Anzahl der möglichen Routen die Fakultät von n. Legen Sie eine neue Datei ufo_routing.py an und verschieben Sie die bereits erstellte Fakultätsfunktion in diese Datei. Rufen Sie fac auf, um im Hauptprogramm die Anzahl der möglichen Routen auf die Konsole auszugeben.

8. Implementieren Sie in ufo_routing.py eine Funktion, die die kürzeste Route findet.

def find_shortest_route(destinations):

Parameter: list destinations : Ziele

Rückgabewert: list : Kürzeste Route, in der alle Ziele angeflogen werden und die danach zu (0, 0) zurückkehrt, d.h. eine Umsortierung von destinations

Wir lösen diese Aufgabe durch das Ausmessen aller Routen. Eine Liste mit allen Routen erhalten Sie durch die Anweisung

```
routes = list(itertools.permutations(destinations))
```

Dazu muss das Modul itertools importiert werden. Wenn Sie die Liste haben, brauchen Sie nur mehr in einer Schleife die Liste zu durchlaufen, mit der Funktion flight_distance_mult die Länge der Route berechnen und sich dabei die kürzeste Route merken.

Für die Ziele (55.0, 20.0), (-116.5, 95.0), (-10.0, -40.0), (-115.0, 95.0) ist die kürzeste Route übrigens die in der Abbildung eingezeichnete Route [(55.0, 20.0), (-115.0, 95.0), (-116.5, 95.0), (-10.0, -40.0)].

Wenn viele Ziele angeflogen werden müssen, ist die Vorgehensweise nicht gut, da die Fakultätsfunktion sehr schnell sehr groß wird. Dieses Problem wollen wir aber nicht vertiefen.

9. Rufen Sie die Funktion find_shortest_route in ufo_main.py an geeigneter Stelle auf:

```
destinations = find_shortest_route(destinations)
```

10. Zu Testzwecken wollen wir zwei weitere alternative Routingstrategien implementieren, auch wenn diese nicht unbedingt die kürzesten Route ergeben. Die beiden Strategien sollen die Ziele folgendermaßen sortieren:

    a) aufsteigend nach ihrer x-Koordinate

    b) aufsteigend nach der Entfernung von (0, 0)

Für beide Strategien müssen wir keine eigenen Sortierfunktionen definieren. Wir können einfach die Python-Sortierfunktion sort auf destinations anwenden. Der Aufruf lautet:

```
destinations.sort(key=key_funktion)
```

Welche key-Funktionen verwendet werden sollen, wird im Folgenden erläutert.

a) Die key-Funktion ist:

```
itemgetter(0)
```

Diese Funktion gibt die erste Komponente eines Tupels zurück. Das ist bei uns die x-Koordinate. Um diese Funktion verwenden zu können, muss die import-Anweisung

```
from operator import itemgetter
```

eingefügt werden.

b) Es muss zuerst eine neue Funktion in der Datei ufo_autopilot.py erstellt werden:

```
def distance_from_zero(p):
Parameter:      tuple p : Punkt (Tupel mit zwei float)
Rückgabewert:   float : Abstand zwischen (0, 0) und p
```

Der Abstand muss nicht neu berechnet werden, sondern es kann die Funktion distance mit den speziellen Parametern aufgerufen werden.

Diese Funktion ist dann die key-Funktion zum Sortieren.

Wir haben jetzt drei Routing-Strategien. Es ist natürlich bei jeder Testausführung immer nur eine aktiv. Die anderen beiden können auskommentiert werden. Testen Sie aber alle drei Strategien. Beobachten Sie dabei die jeweilige Routenlänge.

Praktikum (9)

- Abgabe:
  - Testen Sie das fertige Programm ausgiebig. Wenn alle Tests erfolgreich waren, verpacken Sie die drei py-Dateien ufo_routing.py, ufo_autopilot.py und ufo_main.py in eine zip-Datei. Laden Sie die zip-Datei anschließend in Moodle hoch.
  - Bitte laden Sie die zip-Datei rechtzeitig hoch.
- Checkliste: Überprüfen Sie vor dem Hochladen der Abgabe, ob Folgendes erfüllt ist:
  - Die Abgabe ist eine zip-Datei, die drei py-Dateien enthält.
  - Die py-Datei ufo_autopilot.py enthält die geänderten Funktionen distance, angle, flight_distance, format_flight_data, fly_to, takeoff, cruise, landing und die neuen Funktionen flight_distance_mult, distance_from_zero. Außer diesen Funktionen und benötigten Import-Anweisungen ist kein weiterer Code enthalten.

- Die py-Datei ufo_main.py enthält das geänderte Hauptprogramm. Außer dem Hauptprogramm und benötigten Import-Anweisungen ist kein weiterer Code enthalten.
- Die py-Datei ufo_routing.py enthält die verschobene Funktionen fac und die neue Funktion find_shortest_route. Außer diesen Funktionen und benötigten Import-Anweisungen ist kein weiterer Code enthalten.
- Das Programm hat keine Syntaxfehler.
- Das Programm ist mit verschiedenen Eingabewerten, d.h. Zielelisten, und allen Routing-Strategien getestet und fehlerfrei.