Ufo-Handbuch

Version 3-2-3p

Prof. Dr. Robert Gold

Fakultät Informatik
Technische Hochschule Ingolstadt

Wintersemester 2019/20

Die Klassen UfoSim und UfoView simulieren ein kleines Fluggerät, im Folgenden Ufo genannt. Verwendet werden kann ein Ufo beispielsweise, um Waren in die nähere Umgebung zu liefern. Das Ufo startet dabei in einem fest definierten Punkt. Typischerweise fliegt es verschiedene Zielpunkte an, um danach zum Ausgangspunkt zurückzukehren. Hindernisse, z.B. Hochhäuser oder Funkmasten, sollte es möglichst umfliegen und nicht hineinkrachen.

Der Name Ufo steht für Unified Flying Object, weil damit verschiedene Fluggeräte wie Drohnen, Hubschrauber, Ballone, Raumschiffe der Außerirdischen, etc. simuliert werden können.

1 Flugdaten

Die Position des Ufos wird mit kartesischen Koordinaten (x, y, z) in einem dreidimensionalen Koordinatensystem dargestellt. Der Ursprung des Koordinatensystems ist die anfängliche Startposition des Ufos. Die x-y-Ebene ist die Erdoberfläche. (Wie wir wissen, ist die Erde flach wie eine Scheibe?) Das Ufo selbst ist in der Simulation idealisiert punktförmig.

Sie können sich vorstellen, dass die x-Achse die West-Ost-Richtung und die y-Achse die Süd-Nord-Richtung ist (Abb. 1). Die z-Koordinate ist die Flughöhe (Abb. 2). Eine negative Flughöhe bedeutet einen Absturz des Ufos.

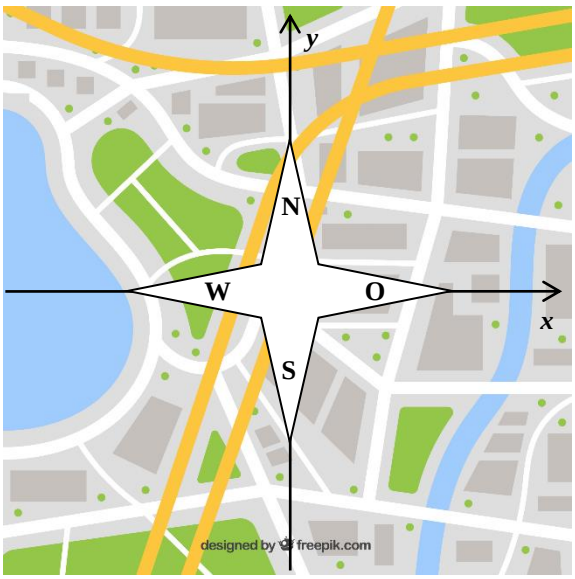

Abb. 1: *x*-*y*-Ebene

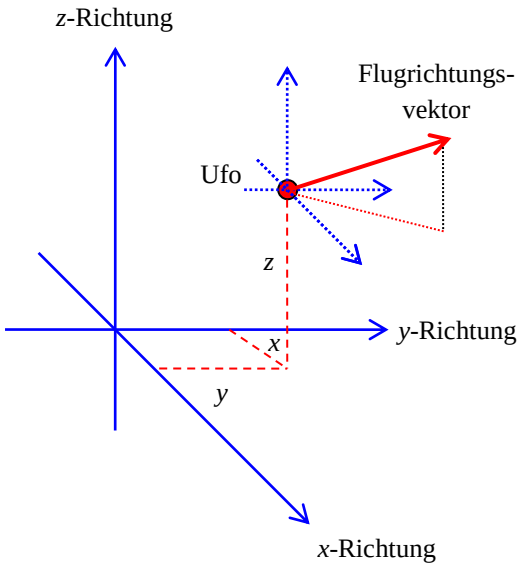

Abb. 2: Dreidimensionale Ansicht

Das Ufo hat außerdem eine Flugrichtung (*d*, *i*) und eine Fluggeschwindigkeit *v* längs der Flugrichtung ($0 \le v \le$ VMAX). Die Richtung *d* ist der Winkel in Grad zwischen der positiven *x*-Achse und der Projektion des Flugrichtungsvektors auf die *x*-*y*-Ebene ($0 \le d \le 359$). Die Steigung *i* ist der Winkel in Grad zwischen der positiven *x*-Achse und der Projektion des Flugrichtungsvektor auf die *x*-*z*-Ebene ($-90 \le i \le 90$). Eine negative Richtung zeigt nach unten (Abb. 3).

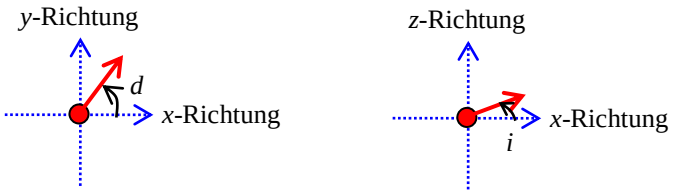

Abb. 3: Projektion auf die *x*-*y*-Ebene bzw. auf die *x*-*z*-Ebene

Beispiele:

$x = 75$, $y = 100$, $z = 197$, $d = 60$, $i = 22$: entspricht ungefähr der Abbildung

$x = 0$, $y = 0$, $z = 0$, $d = 90$, $i = 90$: Startposition im Ursprung, Richtung senkrecht nach oben

$d = 180$, $i = 0$: Das Ufo fliegt in parallel zur *x*-Achse in negativer Richtung.

$i = 90$: Das Ufo steigt senkrecht auf.

$i = –90$: Das Ufo fliegt senkrecht nach unten.

$z \le 0$, $v > 1$: Das Ufo stürzt ab.

$z \le 0$, $v = 1$: Das Ufo landet. Man darf also nur mit Geschwindigkeit 1 landen.

Verringerung von *d*: Das Ufo fliegt nach rechts.

Verringerung von *i*: Das Ufo neigt sich nach unten.

Das Ufo hat also folgende Flugparameter:

Name	Bedeutung	Datentyp	Einheit	Bedingung	Initialwert
x	*x*-Koordinate	Gleitpunktzahl	m		0
y	*y*-Koordinate	Gleitpunktzahl	m		0
z	*z*-Koordinate	Gleitpunktzahl	m		0
v	Geschwindigkeit	Ganze Zahl	km/h	$0 \le v \le$ VMAX	0
d	Richtung	Ganze Zahl	Grad	$0 \le d \le 359$	90
i	Neigung	Ganze Zahl	Grad	$–90 \le i \le 90$	90

Leider wird das Ufo nur simuliert. Der Vorteil davon ist, dass uns das Gerät nicht auf dem Kopf fallen kann. Außerdem haben wir immer schönes Wetter. Wind und Regen haben keinen Einfluss.

VMAX ist zurzeit auf 50 km/h gesetzt. Man kann also recht flott Bahnen über den Himmel ziehen. (Naja, für Außerirdische wäre es wohl eher Schneckentempo.)

Das Ufo misst die seit dem letzten Reset geflogene Distanz (*dist*) längs der Flugrichtung und die dafür benötigte Zeit (*time*). Wenn die Höhe 0 ist, läuft die Zeit nicht weiter.

Das Ufo hat außerdem ein Nahbereichsradar eingebaut (Abb. 4). Dieser Sensor detektiert Hindernisse, die sich im Bereich zwischen 0 und der Konstanten RADAR_RANGE in

Flugrichtung befinden. Der zurückgelieferte Wert *radar* (Gleitpunktzahl, $0 \le radar \le$ RADAR_RANGE) gibt den Abstand des Hindernisses in Flugrichtung an. Falls innerhalb des Messbereiches kein Hindernis gefunden wird, wird –1 zurückgegeben. Das Radar arbeitet nur, wenn das Ufo fliegt, d.h. wenn $z > 0$. Der Boden ist auch ein Hindernis. Das Radar schlägt also auch an, wenn sich das Ufo dem Boden nähert. RADAR_RANGE ist in der aktuellen Version gleich 50 m.

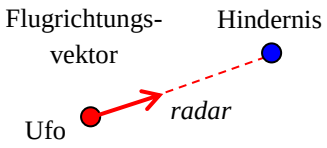

Abb. 4: Radar

Weitere Flugdaten:

Name	Bedeutung	Datentyp	Einheit	Bedingung	Initialwert
dist	Geflogene Strecke	Gleitpunktzahl	m	$dist \ge 0$	0
radar	Distanz zu Hindernis	Gleitpunktzahl	m	$0 \le radar \le$ RADAR_RANGE oder $radar = -1$	–1
time	Flugzeit	Gleitpunktzahl	s	$time \ge 0$	0

2 Auslesen der Flugdaten

Die Flugdaten können durch ein Programm nicht direkt, sondern nur über die folgenden Funktionen ausgelesen werden:

Name	Bedeutung der Funktion	Parameter	Rückgabetyp
getX	Holen der *x*-Koordinate	keiner	Gleitpunktzahl
getY	Holen der *y*-Koordinate	keiner	Gleitpunktzahl
getZ	Holen der *z*-Koordinate	keiner	Gleitpunktzahl
getV	Holen der Geschwindigkeit	keiner	Ganze Zahl
getD	Holen der Richtung	keiner	Ganze Zahl
getI	Holen der Neigung	keiner	Ganze Zahl
getDist	Holen der geflogenen Strecke	keiner	Gleitpunktzahl
getRadar	Holen der Flugzeit	keiner	Gleitpunktzahl
getTime	Holen der Distanz zu Hindernis	keiner	Gleitpunktzahl

Diese Funktionen werden von der Simulationsklasse bereitgestellt und können verwendet werden.

3 Steuerung des Ufos

Das Ufo lässt längs des Flugvektors beschleunigen. Es nur eine konstante Beschleunigung mit dem Beschleunigungswert ACCELERATION möglich. Die Konstante hat den Wert 1 km/h/0.1s, d.h. in einer Sekunde erhöht sich die Geschwindigkeit um 10 km/h. Die Verzögerung geschieht analog mit –ACCELERATION km/h/0.1s.

Da sich das Ufo immer längs des Flugrichtungsvektors bewegt, müssen zum Steuern des Ufos die Richtung und die Neigung verändert werden.

Die Simulationsklasse bietet zur Steuerung folgende Funktionen an.

Name	Bedeutung der Funktion	Parameter	Rückgabetyp
requestDeltaV	Setzen eines Wunsches zur Geschwindigkeitsänderung. Das Ufo beschleunigt bzw. verzögert selbstständig auf den neuen Geschwindigkeitswert. Die Bedingung $0 \leq v \leq$ VMAX wird vom Ufo eingehalten. Beschleunigungs-/Verzögerungswert: ACCELERATION bzw. –ACCELERATION	Ganze Zahl	keiner
requestDeltaD	Setzen eines Wunsches zur Richtungsänderung. Das Ufo dreht sich selbstständig in die neue Richtung. Die Bedingung $0 \leq d \leq 359$ wird vom Ufo eingehalten. Bei einer Linksdrehung über 359 Grad, dreht das Ufo weiter auf 0 Grad, 1 Grad, … Entsprechendes gilt bei einer Rechtsdrehung. Die Drehung erfolgt mit 10 Grad / s.	Ganze Zahl	keiner
requestDeltaI	Setzen eines Wunsches zur Neigungsänderung. Das Ufo neigt sich selbstständig in die neue Richtung. Die Bedingung $-90 \leq i \leq 90$ wird vom Ufo eingehalten. Die Neigung erfolgt mit 10 Grad / s.	Ganze Zahl	keiner
setD	Setzen der neuen Richtung. Die Richtung wird sofort geändert. Dies ist nicht wirklich realistisch, aber praktisch. Falls der Parameter < 0 oder > 359 ist, erfolgt keine Änderung.	Ganze Zahl	keiner
setI	Setzen der neuen Neigung. Die Neigung wird sofort geändert. Dies ist nicht wirklich realistisch, aber praktisch. Falls der Parameter < –90 oder > 90 ist, erfolgt keine Änderung.	Ganze Zahl	keiner

Es ist zu beachten, dass das Ausführen von Änderungswünschen Zeit braucht, z.B. Ändern der Geschwindigkeit von 0 auf 20 km/h: 2 s, Drehen um 180 Grad: 18 s. Es gibt keine verzögerungsfreie Funktion setV. Das wäre zu weit weg von der Realität. Die Verwendung von setD und setI erleichtert die Programmierung sehr.

Beispiele:

vorher	Funktionsaufruf	nachher
$v = 10$	requestDeltaV(20)	Nach 2s ist $v = 30$
$v = 10$	requestDeltaV(–20)	Nach 1s ist $v = 0$
$i = 45$	requestDeltaI(–65)	Nach 6,5s ist $i = –20$
$d = 45$	requestDeltaD(330)	Nach 33s ist $d = 15$
$i = 45$	setI(–20)	Nach 0s ist $i = –20$
$i = 45$	setI(100)	Keine Änderung

Das Ufo regelt seine Antriebe selbstständig so, dass die eingestellte Geschwindigkeit, Richtung und Neigung genau eingehalten werden.

In der Programmentwicklung sind die beiden folgenden Funktionen nützlich.

Name	Bedeutung der Funktion	Parameter	Rückgabetyp
setSpeedup	Normalerweise läuft die Simulation in Echtzeit, d.h. eine Sekunde in der Simulation entspricht einer Sekunde in der Realität. Wem das zu langsam ist, kann den Speedup auf einen Wert zwischen 2 und 50 setzen. Die Änderung des Speedups ist nur einmal möglich. Am besten ganz am Anfang des Programmlaufs.	Ganze Zahl	keiner
reset	Setzt alle Flugdaten auf den jeweiligen Initialwert zurück	keiner	keiner

Zusammenfassung der öffentlichen Konstanten der Simulation:

Name	Bedeutung der Konstante	Datentyp	Einheit
VMAX	Höchstgeschwindigkeit	Ganze Zahl	km/h
RADAR_RANGE	Sichtweite des Radars	Gleitpunktzahl	m
ACCELERATION	Beschleunigung	Ganze Zahl	km/h/0.1s
SPEEDUP	SPEEDUP Sekunden in der Simulation entsprechen einer Sekunde in der Realität. Kann einmal geändert werden	Ganze Zahl	keine

4 Programmierbeispiel

Zum besseren Verständnis der Funktionen der Simulation folgt ein kommentiertes Programmierbeispiele in Python. Das Beispiel fliegt das Ufo von (0, 0, 0) nach (20, 20, 0) (Abb. 5).

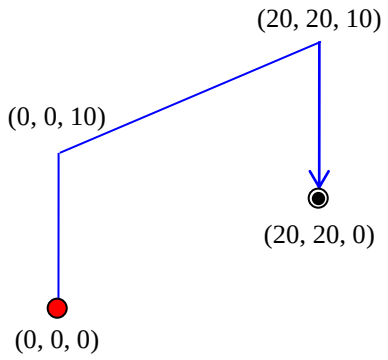

Abb. 5: Beispielflug

Um die Simulation, die in Java geschrieben ist, verwenden zu können, muss das Py4J-Gateway installiert werden. Geben Sie dazu das Kommando pip install py4j in die Kommandozeile ein. Im Python-Programm muss JavaGateway importiert werden, das Gateway initialisiert werden und eine Referenz auf die Simulation geholt werden (siehe Programmierbeispiel). Mehr zu Py4J auf https://www.py4j.org.

ufo_example.py:

```
from py4j.java_gateway import JavaGateway
import math

# Hilfsfunktion zur Formatierung der Zeit und der Koordinaten des Ufos
def format_flight_data(sim):
   return "{0:5.1f} s: [{1:6.1f} {2:6.1f} {3:5.1f}] ".format(
   sim.getTime(), sim.getX(), sim.getY(), sim.getZ())

# Initialisierung des Gateways zur Java-Ufo-Simulation
gateway = JavaGateway()

# In der folgenden Zeile definieren wir eine Referenz auf die Simulation.
sim = gateway.entry_point
sim.reset()

# Doppelte Simulationsgeschwindigkeit
sim.setSpeedup(2)

# Der folgende Code fliegt das Ufo zur Position (20, 20, 0).

# Das Ufo fliegt senkrecht nach oben mit 10 km/h.
print(format_flight_data(sim) + "takeoff with 10 km/h to alt 10 m...")
sim.setI(90)
sim.requestDeltaV(10)
```

```
# Wenn die Hoehe 8m erreicht ist, bremst das Ufo auf 1 km/h.
while sim.getZ() < 8:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h... ")
sim.requestDeltaV(-9)

# Wenn die Hoehe 9.95m erreicht ist, stoppt das Ufo und richtet sich horizontal aus.
while sim.getZ() < 9.95:
   pass
print(format_flight_data(sim) + "...stop and turn horizontal")
sim.requestDeltaV(-1)
sim.setI(0)

# Weiter geht es in Richtung 45 Grad. Die zu fliegende Distanz ist dist.
sim.setD(45)
dist = sim.getDist() + math.sqrt(20 * 20 + 20 * 20)

# Das Ufo beschleunigt auf 15 km/h.
print(format_flight_data(sim) + "go " + str(45) + " deg with 15 km/h...")
sim.requestDeltaV(15)

# Wenn der Abstand zum Ziel 4m ist, bremst das Ufo auf 1 km/h.
while dist - sim.getDist() > 4:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h... ")
sim.requestDeltaV(-14)

# Wenn der Abstand zum Ziel 0.05m ist, stoppt das Ufo.
while dist - sim.getDist() > 0.05:
   pass
print(format_flight_data(sim) + "...stop")
sim.requestDeltaV(-1)

# Das Ufo fliegt senkrecht nach unten mit 10 km/h.
print(format_flight_data(sim) + "landing with 10 km/h")
sim.setI(-90)
sim.requestDeltaV(10)

# Wenn die Hoehe 3m erreicht, bremst das Ufo auf 1 km/h.
while sim.getZ() > 3:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h...")
sim.requestDeltaV(-9)

# Das Ufo ist gelandet, wenn die Höhe kleiner gleich 0 ist.
while sim.getZ() > 0:
   pass
print(format_flight_data(sim) + "...happily landed")
```

Bevor das Python-Skript ausgeführt werden kann, muss die Simulation gestartet werden. Geben Sie dazu in die Kommandozeile den Befehl

java -cp py4j-java_0.10.8.1-bnd-LoVopg.jar;ufosim3-2-3p.jar de.thi.ufo.UfoSim

ein. Beendet wird die Simulation durch Eingabe von Strg+c in die Kommandozeile.

Das Python-Skript kann in einer Entwicklungsumgebung ausgeführt werden oder in der Kommandozeile durch

```
python ufoexample.py
```

Die Simulation muss nicht nach jedem Programmlauf beendet werden, sondern kann im Hintergrund weiterlaufen. Es sollte aber am Anfang des Programmlaufs `reset()` aufgerufen werden, um die Simulation auf die Ausgangswerte zurückzusetzen.

5 Hindernisse

In der aktuellen Version der Simulation bestehen Hindernisse aus einer Menge von achsenparallelen Quadern. Die Anzahl der Hindernisse ist beliebig. Allerdings gibt es aktuell nur ein Hindernis. Fliegt das Ufo in ein Hindernis bleibt die Programmausführung hängen. Probieren Sie es und fliegen Sie von (0, 0, 0) nach (70, 70, 0)!

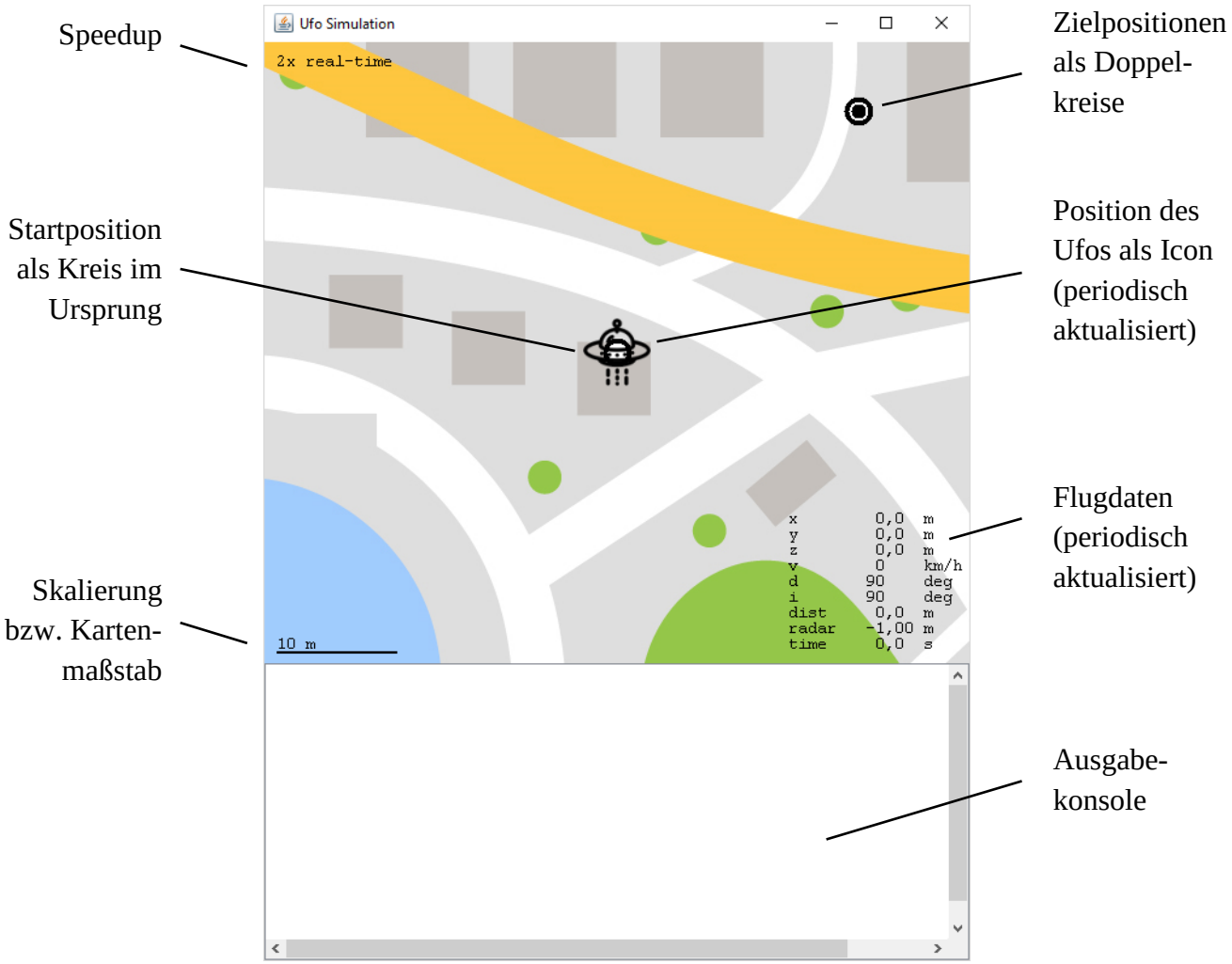

Abb. 6: View nach dem Starten (mit Konsole)

6 View

Ein echtes Fluggerät kann während Testflügen beobachtet werden. Das geht in der Simulation leider nicht. Möglich ist es aber, während des Fluges Meldungen auf die Konsole auszugeben, wie im Programmierbeispiel gezeigt.

Um eine kontinuierliche Beobachtung zu ermöglichen, hat der Ersteller der Simulation eine Klasse UfoView in die Simulation eingebaut.

Die View zeigt verschiedene Daten an (Abb. 6).

Die Karte ist lediglich ein Hintergrundbild. Der Maßstab, d.h. Pixel pro Meter ändert sich bei Verkleinerung oder Vergrößerung des Fensters nicht. Es ist lediglich ein größerer Kartenausschnitt sichtbar.

Im unteren Teil des Fensters befindet sich eine Ausgabekonsole, in der Meldungen angezeigt werden können.

Die Flugdaten, die Position des Ufos und die Ausgaben in der Konsole werden laufend aktualisiert (Abb. 7).

Abb. 7: View während des Fluges (ohne Konsole)

Ist bereits ein View-Fenster bereits geöffnet, kann die Funktion openViewWindow zwar aufgerufen werden, es öffnet sich aber kein zweites View-Fenster. Es werden lediglich die Zielpunkte entfernt.

Die Simulation läuft im Hintergrund weiter, auch wenn kein View-Fenster geöffnet ist.

Die View wird über die Simulation mit den folgenden Funktionen gesteuert:

Name	Bedeutung der Funktion	Parameter	Rückgabetyp
openViewWindow	Öffnet ein View-Fenster, falls keines geöffnet. Der erste Parameter gibt an, ob das Fenster eine Konsole haben soll oder nicht. Ist der zweite Parameter auf wahr gesetzt, bleibt das Fenster immer über allen anderen Fenstern sichtbar. Der dritte Parameter ist die Skalierung, d.h. die Anzahl der Pixel pro m. Falls bereits ein View-Fenster geöffnet ist, werden lediglich die Zielpunkte entfernt.	Zwei boolesche Werte und eine ganze Zahl	Keiner
print	Gibt den Parameter-String auf die Konsole aus. Falls keine Konsole konfiguriert wurde, wird der String auf die Standard-ausgabekonsole geschrieben. In beiden Fällen wird danach ein Zeilenumbruch eingefügt.	String	Keiner
addDestination	Zeichnet einen Zielpunkt in das View-Fenster ein. Die Parameter sind die *x*- und die *y*-Koordinate des Ziels. Es kann beliebig viele Zielpunkte geben.	Zwei Gleitpunkt-zahlen	Keiner

7 Programmierbeispiel

Das python-Beispiel von oben wird im Folgenden durch die Verwendung der View ergänzt.

ufo_example.py:

```
from py4j.java_gateway import JavaGateway
import math

# Hilfsfunktion zur Formatierung der Zeit und der Koordinaten des Ufos
def format_flight_data(sim):
   return "{0:5.1f} s: [{1:6.1f} {2:6.1f} {3:5.1f}] ".format(
   sim.getTime(), sim.getX(), sim.getY(), sim.getZ())

# Initialisierung des Gateways zur Java-Ufo-Simulation
gateway = JavaGateway()

# In der folgenden Zeile definieren wir eine Referenz auf die Simulation.
sim = gateway.entry_point
sim.reset()
```

```
# Oeffnen einer View, die immer on Top angezeigt wird.
# Die Skalierung ist 10 m pro Pixel.
sim.openViewWindow(False, True, 10)

# Ziele des Ufos einzeichnen
sim.addDestination(20.0, 20.0)

# Doppelte Simulationsgeschwindigkeit
sim.setSpeedup(2)

# Meldung auf die Konsole ausgeben und auf Eingabe warten
input("Press return to start...")

# Der folgende Code fliegt das Ufo zur Position (20, 20, 0).

# Das Ufo fliegt senkrecht nach oben mit 10 km/h.
print(format_flight_data(sim) + "takeoff with 10 km/h to alt 10 m...")
sim.setI(90)
sim.requestDeltaV(10)

# Wenn die Hoehe 8m erreicht ist, bremst das Ufo auf 1 km/h.
while sim.getZ() < 8:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h... ")
sim.requestDeltaV(-9)

# Wenn die Hoehe 9.95m erreicht ist, stoppt das Ufo und richtet sich horizontal aus.
while sim.getZ() < 9.95:
   pass
print(format_flight_data(sim) + "...stop and turn horizontal")
sim.requestDeltaV(-1)
sim.setI(0)

# Weiter geht es in Richtung 45 Grad. Die zu fliegende Distanz ist dist.
sim.setD(45)
dist = sim.getDist() + math.sqrt(20 * 20 + 20 * 20)

# Das Ufo beschleunigt auf 15 km/h.
print(format_flight_data(sim) + "go " + str(45) + " deg with 15 km/h...")
sim.requestDeltaV(15)

# Wenn der Abstand zum Ziel 4m ist, bremst das Ufo auf 1 km/h.
while dist - sim.getDist() > 4:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h... ")
sim.requestDeltaV(-14)

# Wenn der Abstand zum Ziel 0.05m ist, stoppt das Ufo.
while dist - sim.getDist() > 0.05:
   pass
print(format_flight_data(sim) + "...stop")
sim.requestDeltaV(-1)
```

```
# Das Ufo fliegt senkrecht nach unten mit 10 km/h.
print(format_flight_data(sim) + "landing with 10 km/h")
sim.setI(-90)
sim.requestDeltaV(10)

# Wenn die Hoehe 3m erreicht, bremst das Ufo auf 1 km/h.
while sim.getZ() > 3:
   pass
print(format_flight_data(sim) + "...slow down to 1 km/h...")
sim.requestDeltaV(-9)

# Das Ufo ist gelandet, wenn die Hoehe kleiner gleich 0 ist.
while sim.getZ() > 0:
   pass
print(format_flight_data(sim) + "...happily landed")
```

Um Ausgaben auf der Ausgabekonsole der View anzuzeigen, muss der erste Parameter von openViewWindow auf True gesetzt werden. Die Ausgabe erfolgt durch Aufruf der Funktion print der Klasse UfoSim, z.B. sim.print(format_flight_data(sim) + "takeoff with 10 km/h to alt 10 m...").

8 Zusammenfassung

der öffentlichen Konstanten und Funktionen der Klasse UfoSim:

Name	Bedeutung der Konstante	Datentyp	Einheit
VMAX	Höchstgeschwindigkeit	Ganze Zahl	km/h
RADAR_RANGE	Sichtweite des Radars	Gleitpunktzahl	m
ACCELERATION	Beschleunigung	Ganze Zahl	km/h/0.1s
SPEEDUP	Echtzeit-Speedup	Ganze Zahl	keine
Name	Bedeutung der Funktion	Parameter	Rückgabetyp
getX	Holen der *x*-Koordinate	keiner	Gleitpunktzahl
getY	Holen der *y*-Koordinate	keiner	Gleitpunktzahl
getZ	Holen der *z*-Koordinate	keiner	Gleitpunktzahl
getV	Holen der Geschwindigkeit	keiner	Ganze Zahl
getD	Holen der Richtung	keiner	Ganze Zahl
getI	Holen der Neigung	keiner	Ganze Zahl
getDist	Holen der geflogenen Strecke	keiner	Gleitpunktzahl
getRadar	Holen der Flugzeit	keiner	Gleitpunktzahl
getTime	Holen der Distanz zu Hindernis	keiner	Gleitpunktzahl
requestDeltaV	Setzen eines Wunsches zur	Ganze Zahl	keiner

	Geschwindigkeitsänderung		
requestDeltaD	Setzen eines Wunsches zur Richtungsänderung	Ganze Zahl	keiner
requestDeltaI	Setzen eines Wunsches zur Neigungsänderung	Ganze Zahl	keiner
setD	Setzen der neuen Richtung	Ganze Zahl	keiner
setI	Setzen der neuen Neigung	Ganze Zahl	keiner
setSpeedup	Änderung des Speedups	Ganze Zahl	keiner
reset	Setzt alle Flugdaten auf den jeweiligen Initialwert zurück	keiner	keiner
openViewWindow	Öffnet ein View-Fenster	Zwei boolesche Werte und eine ganze Zahl	Keiner
print	Gibt den Parameter-String auf die Konsole aus	String	Keiner
addDestinations	Zeichnet einen Zielpunkt ein	Zwei Gleitpunkt-zahlen	Keiner