Bonus1

Dr. Günter Kolousek

21. Juli 2015

Lege wiederum ein Verzeichnis an. Nennes es `19_bonus1`! In diesem Verzeichnis sollen alle Dateien der jeweiligen Einheit abgelegt werden.

Programmiere ein Mondlandungsspiel!

Entwickle eine einfache Simulation einer Mondlandung. Nimm an die Raumfähre befindet sich in 50.000m Höhe über der Mondoberfläche und hat eine Abwärtsgeschwindigkeit von derzeit 3600km/h. Wenn kein Schub gegeben wird, dann wird die Raumfähre mit $1.63 m/s^2$ vom Mond angezogen.

Die Antriebe der Raumfähre können bei vollem Schub jedoch eine Gegenbeschleunigung von $13.63 m/s^2$ erzeugen und verbrauchen dabei pro Sekunde 100 Liter Treibstoff. Anfangs verfügt die Raumfähre über 10.000 Liter Treibstoff.

Die Raumfähre kann eine Aufsetzgeschwindigkeit von 10m/s aushalten, ansonsten ist das Landemanöver gescheitert. Hierfür ist eine entsprechende Meldung auszugeben.

1. Lege Namen für alle Parameter an.

   a) für alle Anfangsparameter von Geschwindigkeit und Abstand zur Mondoberfläche (Weg) an: `v0` und `s0` an.

   b) für die aktuellen Werte von Zeit, Treibstoff und Schub: `t`, `fuel`, `thrust`. Die Antriebe (bewirken Gegenschub) können entweder ausgeschalten (0) oder eingeschalten (1) sein.

   c) Die Simulation findet in kleinen zeitlichen Schritten statt. Lege einen Namen `delta_t` an. Dafür ist 0.1 ein vernünftiger Anfangswert.

2. Erstelle eine Endlosschleife, die vorerst alle relevanten Parameter nacheinander berechnet. Je Schleifendurchgang findet ein Simulationsschritt mit den Berechnungen der Parameter statt.

a) Dazu soll zuerst die Zeit `delta_t` gewartet werden. Das kannst du mittels Funktion `sleep` aus dem Modul `time` erreicht werden. Diese Funktion wartet die angegebene Zeit und erst nach Ablauf dieser Zeit wird die nächste Anweisung im Programm abgearbeitet.

   D.h. damit tun wir so, als ob in Wirklichkeit die Zeit `delta_t` vergangen ist.

b) Berechne zuerst die aktuelle Beschleunigung, die auf die Raumfähre wirkt. Einerseits wirkt die Anziehungskraft des Mondes auf die Raumfähre und gegenteilig wirkt die Schubkraft der Antriebe (falls diese eingeschalten sind).

   Eingeschalten sind die Antriebe, wenn `thrust` auf 1 gesetzt ist. Wenn die Antriebe nicht eingeschalten sind, dann hat die Raumfähre eine Beschleunigung von 1.63m/s^2.

c) Berechne als nächstes die neuen Treibstoffvorräte. D.h. in Abhängigkeit des Schubes kommt es entweder zu einer Verringerung der Treibstoffvorräte im angegebenen Maße oder diese bleiben gleich.

d) Berechne die neue Geschwindigkeit `v`. Prinzipiell lautet die Formel zur Berechnung der Geschwindigkeit einer gleichmäßig beschleunigten Bewegung v = a · t.

   Bei a handelt es sich um die aktuelle (schon berechnete) Beschleunigung und bei t um die Zeit, die in diesem Simulationsschritt vergangen ist (`delta_t`).

   Allerdings darf nicht vergessen werden dazu die Anfangsgeschwindigkeit `v0` zu addieren!

e) Berechne den neuen Abstand zur Mondoberfläche und lege diesen unter dem Namen `s` ab. Die Formel zur Berechnung des zurückgelegten Weges bei einer gleichmäßig beschleunigten Bewegung lautet $s = v \cdot t + \frac{a}{2} \cdot t^2$.

   Damit kann man ausgehend vom ursprünglichen Abstand `s0` den aktuellen Abstand `s` zur Mondoberfläche berechnen.

f) Schreibe eine Funktion `output_state(v, s, a, t, fuel, thrust)`, die diese Parameter auf der Standardausgabe ausgibt. Z.B. so:

```
Geschwindigkeit: 1000m/s
Höhe: 50000m
a: 1.63m/s2
t: 0s
fuel: 10000L
thrust: 0
```

   Rufe diese Funktion jetzt in der Schleife nach den Berechnungen auf.

g) Jetzt sind noch die Namen `v0` und `s0` mit den errechneten Werten `v` und `s` für den nächsten Simulationsschritt zu setzen.

Außerdem muss noch die Zeit `t` "weitergestellt" werden. Es ist ja mittlerweile schon die Zeit in der Länge von `delta_t` verstrichen.

h) Testen! Es sollte sich das Raumschiff mit immer größerer Geschwindigkeit Richtung Mondoberfläche bewegen! Abbrechen kann man das laufende Programm jederzeit mit `CTRL-C`.

3. Die Simulation funktioniert derzeit nicht so schlecht. Allerdings bricht diese auch nicht ab, wenn das Raumschiff ungebremst in den Mond rast!

   Nichts einfacher als das: Ist die Höhe kleiner oder gleich 0, dann ist das Programm zu beenden (indem einfach die Schleife beendet wird).

   Füge die entsprechende Anweisung gleich direkt nach der Berechnung von `a`, `v` und `s` ein.

4. Als nächsten Schritt wollen wir jetzt noch die Steuerung des Schubes programmieren. Der Benutzer soll den Schub einschalten und auch wieder ausschalten können.

   Das Problem ist allerdings, dass so etwas in einer Kommandozeile nicht so einfach geht. Wir wissen ja schon, dass wir mittels `input` zwar dem Benutzer zwar eine Eingabe machen lassen können, aber dies für so ein Spiel nicht besonders taugt. Einerseits muss der Benutzer bei jeder Eingabe auch die `<Enter>` oder `<Return>` Taste drücken und andererseits wartet unser Programm ja auf die Eingabe des Benutzers und kann somit nicht unsere Endlosschleife gemäß unseren zeitlichen Vorstellungen (`delta_t`) durchlaufen.

   Deshalb greifen wir auf unsere alte Bekannte die `turtle` zurück: Diese erlaubt die einfache Programmierung von ereignisgesteuerten Programmen.

   a) Zuerst schreiben wir eine Funktion `switchon_thrust()`, die die globale Variable `thrust` auf `1` setzt. Beachte:

      - Um eine globale Variable zu verändern (Achtung Pfui!), muss man die `global` Anweisung verwenden.
      - Die globale Variable `thrust` geht direkt in die Berechnung der Beschleunigung der Raumfähre ein.

   b) Schreibe jetzt eine analoge Funktion `switchoff_thrust()`, die den Schub wieder ausschaltet.

      Das ist soweit in Ordnung aber diese Funktionen werden noch überhaupt nicht aufgerufen. Wann sollen die den aufgerufen werden? Wenn der Benutzer bestimmte Tasten drückt. Dazu brauchen wir die Turtle! Auf zum nächsten Punkt.

c) Schreibe eine Funktion `init_turtle()`, die folgende Aktionen durchführt:

   i. sich mit der Funktion `getscreen()` ein `Screen`Objekt holt und dieses unter dem Namen `screen` ablegt.

   ii. Wir legen fest, dass durch Drücken der Taste `1` der Schub eingeschalten werden soll und durch Drücken der Taste `0` dieser wieder ausgeschalten werden soll.

   Das `Screen`-Objekt kennt einige Methoden. Wir benötigen dazu vorerst die `onkey(fun, key)` Methode die sich als ersten Parameter eine Funktion und als zweiten Parameter eine Tastenangabe erwartet.

   D.h. das könnte in unserem Fall für das Einschalten des Schubes folgendermaßen aussehen:

   ```
   screen.onkey(switchon_thrust, "1")
   ```

   Das Ausschalten des Schubes ist analog zu programmieren.

   iii. Zu guter Letzt muss noch dem Turtle-System mitgeteilt werden, dass es jetzt auf die Benutzereingaben hören muss. Es ist die Methode `listen()` auf das `screen` Objekt aufzurufen.

   iv. Ok, zu allerletzt: Nicht vergessen, dass die Funktion auch aufgerufen gehört.

Jetzt sollte sich unser Mondlandespiel schon prinzipiell spielen lassen. Testen!

5. Jetzt müssen wir unbedingt noch überprüfen, ob wir eigentlich richtig gelandet sind. Wann war die Landung erfolgreich? Wenn die Landegeschwindigkeit kleiner oder gleich 10m/s betragen hat.

   a) D.h. innerhalb der Abfrage, ob die Raumfähre schon in die Mondoberfläche gerast ist, ist noch die Geschwindigkeit zu überprüfen und jeweils eine entsprechende Meldung auszugeben.

   Möglich wäre z.B. "Ship landed" oder "Boom" (je nach dem)…

   b) Soweit ist das ja gar nicht so schlecht, nur leider nicht ganz korrekt, da die errechnete Geschwindigkeit die Geschwindigkeit ist, die nach Ablauf der Zeitspanne `delta_t` aufgetreten ist. Die Fähre hat allerdings schon vor Ablauf der Zeitspanne aufgesetzt (in welcher Form auch immer) und daher zu diesem Zeitpunkt unter Umständen eine niedrigere Geschwindigkeit als berechnet.

   Tja, jetzt wird es allerdings etwas kompliziert. Deshalb folgt hier direkt der Programmcode zur Berechnung der exakten Geschwindigkeit beim Auftreffen auf die Mondoberfläche:

```
# Berechnung der beiden Lösungen einer quadratischen Gleichung
t1_1 = -v0 / a + math.sqrt((v0 / a) ** 2 + 2 * s0 / a)
t1_2 = -v0 / a - math.sqrt((v0 / a) ** 2 + 2 * s0 / a)

if t1_1 <= delta_t:
    t1 = t1_1
else:
    t1 = t1_2

v1 = v0 + t1 * a
```

Baue diese Anweisungen einfach in das Programm ein. In `v1` befindet sich die korrekte Geschwindigkeit, die zu überprüfen ist!

Auf der Standardausgabe ist jetzt die tatsächliche Landegeschwindigkeit auszugeben, da diese interessant ist.

6. Die Ausgabe auf der Standardausgabe ist eigentlich auch nicht so richtig nett. Im Turtle-Fenster wäre das schon besser. Los geht's! Die `write` Methode hilft. Die Turtle braucht auch keiner sehen.

   Weiters reichen jetzt die Werte der Geschwindigkeit, die Höhe und der Treibstoffvorrat und der Schub. Alle Werte sollen gerundet auf 0 Stellen ausgegeben werden (die Nachkommastellen interessieren in diesem Fall einfach nicht).

7. Um das lästige "Flackern" der Ausgabe abzuschalten ist in der `init_turtle` Methode folgende Anweisung hinzuzufügen: `tracer(0)`.

8. Im Moment hat unser Raumschiff noch unbegrenzt Treibstoff zur Verfügung, da wir den aktuellen Stand noch nicht überprüft haben.

   Bauen wir dazu die `switchon_thrust` Funktion so um, dass wenn der Treibstoff

   a) kleiner als 0.1 ist, sowohl der Treibstoffvorrat auf 0 als auch der Schub auf 0 gesetzt werden.

   b) wenn anderenfalls der Treibstoffvorrat kleiner als 100 ist, nur mehr ein entsprechender Anteil an Schub zur Verfügung gestellt wird (also ein Wert kleiner als 0) errechnet wird. Wie?

   c) anderenfalls der Schub auf 1 gesetzt wird.