Kapitel 6
Das Spiel »Snake«

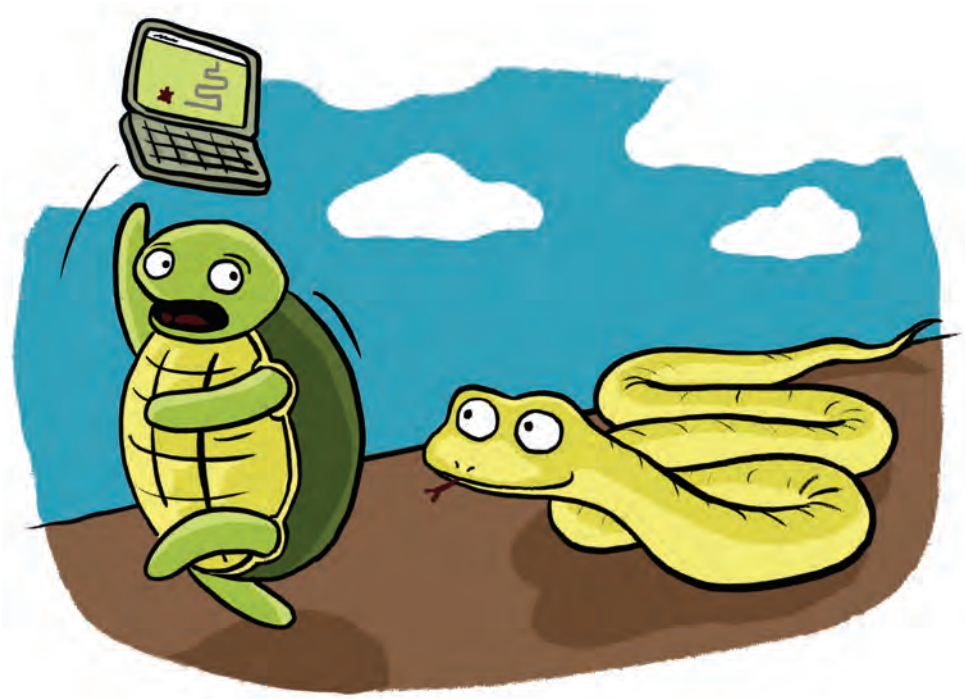


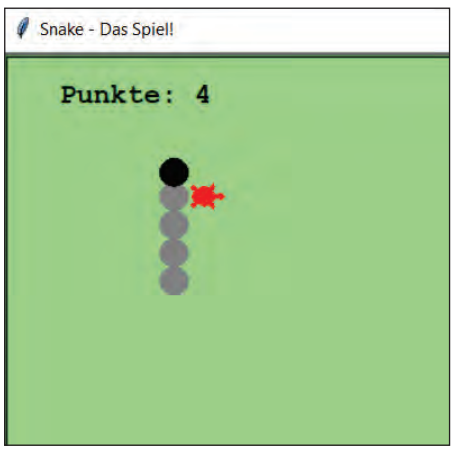
Illustration: Leonard Ermel

Wer viele Muster und Bilder malt, muss auch mal ausruhen und entspannen. Und was ist entspannender als ein schönes, klassisches Computerspiel? In diesem Kapitel erfährst du, wie du mit Schildkröten nicht nur zeichnen, sondern auch spielen kannst. Und warum sich Schildkröten besser vor (Python-)Schlangen hüten sollten . . .

Von Schildkröten zu Schlangen

In klassischen Computerspielen wird eine Spielfigur über den Bildschirm bewegt, löst Aufgaben, sammelt Punkte, verliert Leben – und nach einer bestimmten Zeit oder bei einem bestimmten Ereignis ist das Spiel zu Ende. So ist es auch

beim beliebten *Snake*-Spiel (auf Deutsch »Schlange«, auch bekannt unter dem Namen *Wurm* oder *Pizza-Wurm*), wo eine Schlange auf der Suche nach Futter über den Bildschirm wandert und dabei immer länger wird.



Was liegt näher, als das Snake-Spiel in Python zu programmieren und das turtle-Modul zu nutzen, um Schlangen Schildkröten fressen zu lassen . . .

Die Spielregeln

Hier kommen die Spielregeln, falls du noch nie *Snake* gespielt hast:

- Du steuerst die Schlange mithilfe der Pfeiltasten zu der Stelle auf dem Spielfeld, wo Futter liegt.
- Berührt der Kopf der Schlange das Futter, so wächst die Schlange hinten ein Stück, und du bekommst einen Punkt. Das Futter verschwindet, und an einer anderen Stelle taucht neues Futter auf.
- Weil die Schlange mit jeder Futterportion, die sie frisst, immer länger wird, wird das Spiel immer schwieriger. Denn du darfst mit dem Schlangenkopf weder den Spielfeldrand noch den eigenen Schlangenkörper berühren. Sobald das passiert, ist das Spiel zu Ende und die Schlange schrumpft wieder auf ihre Anfangsgröße.

Das Design des Spiels

Im *Snake*-Spiel brauchst du folgende Dinge:

» das `Spielfeld`: Das ist ein Fenster auf dem Bildschirm, wo du die Spielfiguren (*Objekte*) anzeigen und bewegen kannst. Dieses Fenster wird so erstellt:

```
Spielfeld = Screen()
```

» die Spielfiguren: Das sind die Objekte auf dem Spielfeld, also die Schlange, das Futter und die Punktestandsanzeige.

Deine Schlange `Snake` ist eigentlich eine Schildkröte (ein `Turtle()`-Objekt) und wird daher folgendermaßen erstellt:

```
Snake = Turtle()
```

Spielfeld und Spielfiguren kannst du zu Beginn des Spiels erzeugen und durch Aufrufen entsprechender Funktionen *initialisieren*, also die Anfangseigenschaften festlegen. Hier ist die Funktion zur Initialisierung des Spielfelds:

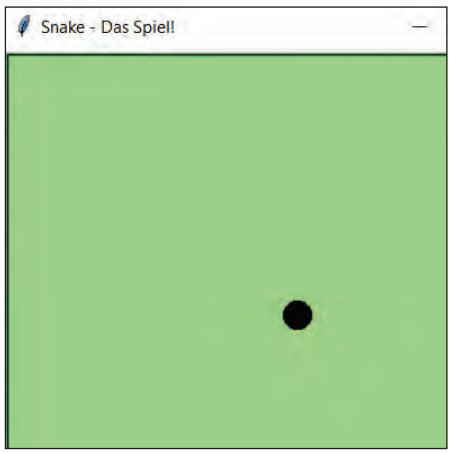
```
def initSpielfeld(Breite, Hoehe):
   Spielfeld.title("Snake - Das Spiel!")  # Titelzeile
   Spielfeld.bgcolor("lightgreen")        # Hintergrundfarbe
   Spielfeld.setup(Breite, Hoehe)         # Größe
```

Und hier kommt die Funktion zur Initialisierung der Schlange:

```
def initSchlange():
   Snake.speed(1)           # Die Schlange ist langsam,
   Snake.shape("circle")    # der Kopf ist ein Kreis
   Snake.color("black")     # und schwarz.
   Snake.richtung = "keine" # Sie hat zunächst keine Richtung,
   Snake.penup()            # sie hinterlässt keine Spur,
   Snake.goto(0, 100)       # und sie startet an Position (0, 100).
```

Hier ist das erste Programmgerüst, das nichts weiter macht als das Spielfeld mit einer minimal kurzen Anfangsschlange darauf anzuzeigen, die eigentlich nur aus einem Kopf besteht. Futter gibt es noch nicht.

```
Spielfeld = Screen()    # Spielfeld wird erzeugt.
Snake = Turtle()        # Schlange Snake wird erzeugt.
initSpielfeld(480, 480) # Spielfeld bekommt Anfangseigenschaften
initSchlange()          # Snake bekommt Anfangseigenschaften
Spielfeld.update()      # Zeige alles an
```



Bekommst du das hin? *Probiere das Programm aus und variiere die Design-Eigenschaften. Mache das Spielfeld rechteckig und blau und den Schlangenkopf rot.*

Den Programmcode findest du in der Datei Snake1_Design.py.

Es kommt Bewegung in die Schlange

Als Nächstes soll sich die Schlange bewegen, und zwar immer in die Richtung, die der gerade gedrückten Pfeiltaste entspricht.

Auf Tastendruck-Ereignisse lauschen

Du kannst ausnutzen, dass jeder Tastendruck ein sogenanntes *Event* auslöst: ein Ereignis, das das Spielfeld zur Kenntnis nimmt, wenn wir es dazu auffordern, auf diese Ereignisse zu achten (zu lauschen, auf Englisch *to listen*).

Mit dem Kommando `Spielfeld.listen()` sagst du dem Spielfeld, dass es auf Tastendruck-Events lauschen soll.

Die vier Pfeiltasten ↑, ↓, → und ← lösen die vier folgenden Events aus: `Up`, `Down`, `Right` und `Left` (hoch, runter, rechts, links).

Für jedes dieser Events denkst du dir nun einen Funktionsnamen aus. Zum Beispiel bekommt die Funktion, die beim Eintreten des Events `Up` aufgerufen werden soll, den Namen `gehe_hoch()`. Die Verbindung zwischen Funktionsnamen und Event stellst du durch das Kommando `onkey` her, das einmal für jede

Pfeiltaste in der Initialisierungsfunktion `initRichtung()` aufgerufen wird. Die Funktion `initRichtung()` wird, wie die anderen Initialisierungsfunktionen, beim Programmstart ausgeführt.

```
def initRichtung():
   Spielfeld.listen()   # das Spielfeld lauscht auf Pfeiltasten
   Spielfeld.onkey(gehe_hoch, "Up")
   Spielfeld.onkey(gehe_runter, "Down")
   Spielfeld.onkey(gehe_rechts, "Right")
   Spielfeld.onkey(gehe_links, "Left")
```

Nun fehlen aber noch die Funktionen selbst, also der Code zu `gehe_hoch()`, `gehe_runter()`, `gehe_rechts()` und `gehe_links()`. Hier passiert zunächst nicht viel, außer dass die Richtung der Schlange entsprechend der gedrückten Pfeiltasten gesetzt wird:

```
def gehe_hoch():
   Snake.richtung = "hoch"
def gehe_runter():
   Snake.richtung = "runter"
def gehe_rechts():
   Snake.richtung = "rechts"
def gehe_links():
   Snake.richtung = "links"
```

Und sie bewegt sich doch

Damit sich die Schlange nun aber wirklich ein Stück in die entsprechende Richtung bewegt, muss sich ihre Position verändern. Eine Position wird durch zwei Zahlen angegeben, die x-Koordinate (die sich ändert bei der Bewegung von links nach rechts) und die y-Koordinate (die sich ändert bei der Bewegung von oben nach unten).

Soll die Schlange also einen Schritt nach oben machen (`Snake.richtung` hat den Wert `"hoch"`), dann muss sich ihre y-Koordinate ändern: Sie wird um eine Schrittlänge hochgezählt. Soll sie einen Schritt nach links machen, so muss der Wert der x-Koordinate um eine Schrittlänge heruntergezählt werden.

Du merkst dir also zunächst die aktuelle x-Koordinate und y-Koordinate der Schlange in zwei Variablen: `x = Snake.xcor()` und `y = Snake.ycor()`.

Je nachdem, welche Richtung gewählt wurde, setzt du nun die entsprechende Koordinate der Schlange neu, und zwar auf den Wert, der herauskommt, wenn du eine Schrittlänge (beispielsweise 20 Pixel) zum bisherigen Koordinatenwert addierst oder von ihm abziehst.

Bei `"hoch"` und `"runter"` ändert sich die y-Koordinate, bei `"links"` und `"rechts"` ändert sich die x-Koordinate. Alle vier Fälle kannst du in einer Funktion `bewegeSchlange()` zusammenfassen:

```
def bewegeSchlange():
   x = Snake.xcor()   # (x,y): Position der Schlange
   y = Snake.ycor()
   if Snake.richtung == "hoch":   # Gehe nach oben
      Snake.sety(y+20)
   if Snake.richtung == "runter": # Gehe nach unten
      Snake.sety(y-20)
   if Snake.richtung == "rechts": # Gehe nach rechts
      Snake.setx(x+20)
   if Snake.richtung == "links":  # Gehe nach links
      Snake.setx(x-20)
```

Da man als Spielerin oder Spieler natürlich nicht nur einmal eine Pfeiltaste drückt, sondern immer wieder, und das schnell hintereinander und in ziemlich schnellem Wechsel, musst du im Programm immer wieder die Richtung erkunden und die Schlange entsprechend wandern lassen.

Du brauchst also eine Schleife, in der du dann deine Funktion `bewegeSchlange()` immer wieder aufrufen kannst.

Und nicht nur die, du musst auch das Spielfeld mit der Schlange an neuer Position immer wieder neu zeichnen. Die Funktion zum Spielfeld-neu-zeichnen kennst du schon, sie heißt `Spielfeld.update()`.

Die Schleife soll diese Anweisungen immer wieder aufrufen, und zwar solange das Programm läuft. Das geht mit einer `while`-Schleife. Die Bedingung (das Programm läuft) wird einfach durch den Wahrheitswert `True` ausgedrückt. Denn der ist nun einmal immer wahr (solange das Programm läuft).

So sieht also dein Programm aus (wenn du alle Funktionen davor definiert hast):

```
Spielfeld = Screen()
Snake = Turtle()
initSpielfeld(480, 480)  # (Breite, Höhe)
initSchlange()
initRichtung()
while True:
   Spielfeld.update()
   bewegeSchlange()
```

Bekommst du das hin? *Sorge dafür, dass die Schlange beim Drücken der Leertaste anhält.*

Tipp: Erweitere die Funktion `initRichtung()` *um ein Event-Funktionsnamen-Paar: Das Event, das durch Drücken der Leertaste ausgelöst wird, heißt* `"space"`.

Den Funktionsnamen kannst du dir ausdenken, zum Beispiel `stoppe` *oder* `halte`.

Wenn du dann die Funktion definierst, setzt du die Richtung auf `"keine"`. *Die haben wir schon mal verwendet, und zwar bei der Initialisierung der Schlangeneigenschaften in der Funktion* `init_Schlange()`.

Die Funktion `bewegeSchlange()` *brauchst du übrigens nicht zu erweitern, da sich die Position der Schlange beim Drücken der Leertaste nicht verändert.*

Den Code findest du in der Datei Snake2_Bewegung.py.

Bewegung macht hungrig: Die Schlange sucht Futter

Die Schlange kann nun über das Spielfeld bewegt werden (sogar über die Spielfeldränder hinaus, aber darum kümmern wir uns später). Da ist klar, dass sie irgendwann hungrig wird und Futter braucht. Unter »Futter« verstehen wir Objekte, die auf dem Spielfeld erscheinen. Die Schlange soll dann das Futter »fressen«, indem sie auf die Futterposition gesteuert wird.

Wo kommt das Futter her?

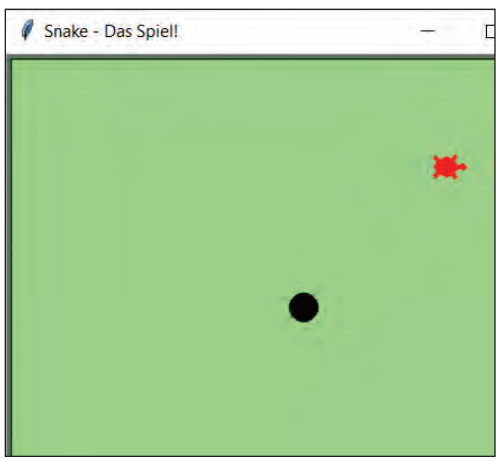
Auch Futter ist ein `Turtle()`-Objekt im Spiel, so wie der Schlangenkopf. Zu Beginn des Programms wird es mit `Futter = Turtle()` erstellt. Wie bei der Schlange gibt es eine Funktion `initFutter()`, mit der die Eigenschaften vom Futter (Farbe, Aussehen, Position) festgelegt werden:

```
def initFutter():
   Futter.shape("square")   # Eine Futterportion ist quadratisch
   Futter.color("red")      # und rot.
   Futter.penup()           # Sie hinterlässt keine Spur
   Futter.goto(100, 100)    # und liegt an Position (100,100).
```

Bekommst du das hin? *Erstelle zu Beginn deines Programms das Futter-Objekt, ergänze die Funktion* `initFutter()` *und rufe sie im Programm auf.*

Das Futter erscheint.

Wenn du möchtest, dass deine Schlange Schildkröten frisst (anstatt Quadrate), dann ändere die Futterform in deiner Funktion `initFutter()` einfach zu `Futter.shape("turtle")`.



In der Realität fressen Schlangen übrigens keine Schildkröten. Aber seit wann sind Computerspiele realistisch?

Das Futter wird gefressen

Nun ist das Futter zwar zu sehen, aber es verschwindet nicht, wenn die Schlange es frisst, also der Schlangenkopf an die Futterposition gesteuert wird. Wie bekommst du das hin?

In der `while-True`-Schleife des Spiels, in der die Schlange immer wieder neu positioniert wird, muss nun auch geprüft werden, ob die aktuelle Schlangenposition mit der Futterposition übereinstimmt. Sollte das der Fall sein, so kannst du das Futter mit folgendem Code verschwinden lassen:

```
if Snake.position() == Futter.position():
   Futter.hideturtle()
```

Jetzt kannst du schon bis zum ersten Futter-Happen spielen!

Neues Futter erscheint ganz woanders

Aber eigentlich soll das Futter ja nicht wirklich verschwinden, sondern irgendwo anders auf dem Spielfeld neu erscheinen. Dafür brauchst du Zufallszahlen, die du schon aus den Kapiteln 3, 4 und 5 kennst. Dort haben wir Zufallszahlen genutzt, um bunte Schildkrötenbilder zu malen.

Für Zufallszahlen musst du ganz oben im Programmcode ein zweites Modul importieren, das `random`-Modul. Ergänze also den Import in deinem Programm mit der Zeile:

```
from random import *
```

Angenommen, die neue Futterposition hat die Koordinaten `fx` und `fy`. Dann kannst du mit `Futter.setposition(fx,fy)` die Futterkoordinaten neu setzen. Aber wie bekommst du passende zufällige Zahlen für `fx` und `fy`?

Hier hilft dir die Funktion `randrange(untereGrenze, obereGrenze, Schrittweite)`.

Sie gibt dir eine Zufallszahl zwischen den beiden Werten, die du für `untereGrenze` und `obereGrenze` einsetzt. Die Zahl `randrange(-4, 8, 2)` ist also eine zufällige Zahl zwischen `-4` und `8`.

Außerdem ist sie nur in 2er-Schritten von der unteren Grenze aus erreichbar, so wie beispielsweise –2 oder 6 (aber nicht 5). Die Zahl `Schrittweite` ist wichtig, da die Schlange sich in 20-Pixel-Schritten bewegt. Würde das Futter dazwischen auftauchen, könnte es die Schlange niemals erreichen. Wir brauchen für die Futterposition also ebenfalls eine Schrittweite von 20 Pixeln, damit sich Schlange und Futter auf denselben Feldern bewegen. Beachte auch, dass die Zahl `untereGrenze` durch 20 teilbar sein muss, damit Schlange und Futter auf denselben Feldern unterwegs sind.

Bekommst du das hin? *Verstecke das Futter nicht mehr mit* `hideturtle()`, *sondern ändere seine Position, indem du mit* `Futter.setposition(fx,fy)` *die Futterkoordinaten neu setzt. Wähle für* `fx` *und* `fy` *passende Zufallszahlen. Nutze dafür die Funktion* `randrange`.

Tipp: *Die Zufallszahl* `fx` *liegt zwischen dem linken Spielfeldrand (untere Grenze) und dem rechten Rand (obere Grenze). Du kennst ja die Breite deines Spielfelds. Da der Mittelpunkt des Spielfelds die*

x-Koordinate 0 hat, erhältst du die x-Koordinate vom linken Rand, indem du von 0 die halbe Spielfeldbreite abziehst. Den x-Wert für den rechten Rand erhältst du, indem du zu 0 die halbe Breite addierst.

Die Zufallszahl `fy` *liegt zwischen dem unteren Spielfeldrand (untere Grenze) und dem oberen Spielfeldrand (obere Grenze). Du kannst die y-Koordinaten des oberen und des unteren Spielfeldrandes anhand der Spielfeldhöhe berechnen.*

Zur Sicherheit nimm für die untere und obere Grenze nicht genau den Rand, sondern eine Zahl etwas weiter innerhalb des Spielfelds, damit das Futter immer sichtbar bleibt.

Hier ist die `while-True`*-Schleife mit Futter-Neupositionierung für ein Spielfeld mit einer Breite von 600 und einer Höhe von 540 Pixeln:*

```
while True:
   bewegeSchlange()
   Spielfeld.update()
   if Snake.position() == Futter.position(): # Futter gefunden
      fx = randrange(-280,280,20)   # neue Zufallsposition
      fy = randrange(-260,260,20)
      Futter.setposition(fx,fy)
```

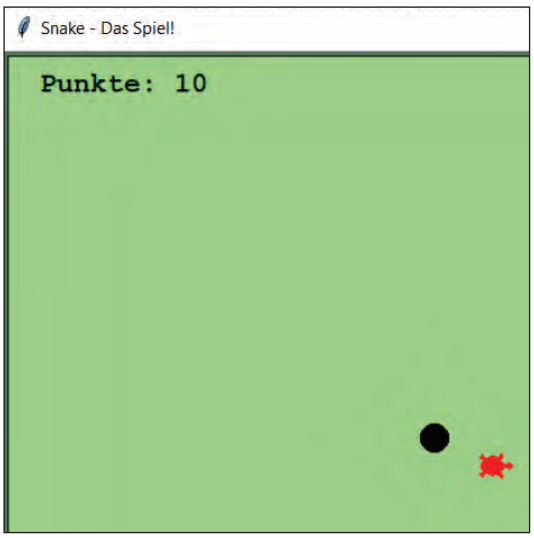
Fressen bringt Punkte

Das Ziel des Spiels (außer Spaß zu haben) ist, dass die Schlange möglichst viel Futter frisst und damit viele Punkte sammelt. Jede Futterportion bringt einen Punkt.

Die aktuelle Punkte-Zahl merkst du dir am besten in einer Variablen, die du `Punkte` nennen könntest und die bei jedem Fressvorgang um 1 erhöht wird. Zu Beginn des Spiels hat diese Variable den Wert 0.

In der `while-True`-Schleife wird der Wert immer dann erhöht, wenn Futter gefunden wird.

So weit, so einfach. Aber der Punktestand soll natürlich auch angezeigt werden. Wie geht das?



Auch die Anzeige ist wieder ein `Turtle()`-Objekt. Du erzeugst sie zu Beginn des Programms wie die Schlange und das Futter mit `Anzeige = Turtle()`. Und auch hier legst du in einer Initialisierungsfunktion die Eigenschaften der Anzeige fest.

```
def initPunkteAnzeige():
   Anzeige.color("black")     # Der Text ist schwarz
   Anzeige.hideturtle()       # das Turtlesymbol ist unsichtbar
   Anzeige.penup()   # die Anzeige-Turtle hinterlässt keine Spur
   Anzeige.setposition(-Breite/2+40, Hoehe/2-40)  # links oben
   Anzeige.write("Punkte: "+str(Punkte),font=("Courier",14,"bold"))
```

Als Position für die Punkteanzeige wurde hier die linke obere Ecke des Spielfelds gewählt. Der Einfachheit halber sind Breite und Höhe des Feldes jetzt in zwei Variablen gespeichert. Die linke obere Spielfeldecke hat die Koordinaten `(-Breite/2, Hoehe/2)`. Die eigentliche Anzeige-Stelle liegt ein wenig (40 Pixel) rechts und unterhalb der Ecke, damit man alles gut lesen kann.

Die Anzeige besteht aus dem Text `Punkte:` und einem Leerzeichen. Das geben wir einfach vor, indem wir beides in Anführungszeichen setzen: `"Punkte: "`. Es folgt der Wert der Variablen `Punkte`, den wir für die Anzeige mit `str()` in Text (auf Englisch *string*) umwandeln.

In einem Programm müssen Variablen zuerst mit Werten belegt werden, bevor sie in Funktionen verwendet werden können.

Bekommst du das hin? *Erstelle zu Beginn deines Programms das Anzeige-Objekt, setze eine* `Punkte`*-Variable auf 0, ergänze die Funktion* `initPunkteAnzeige()` *und rufe sie vor der* `while-True`*-Schleife im Programm auf. Teste, ob die Anzeige sichtbar und gut positioniert ist.*

Ändere das Programm so, dass sie in der rechten unteren Ecke erscheint.

Anzeige der Punkte aktualisieren

Nun fehlt nur noch das Aktualisieren der Anzeige, wenn ein Punkt gewonnen wurde. Dazu wird im Code des Spiels (innerhalb der `while-True`-Schleife) immer dann, wenn Futter gefunden wird, der Wert der Variablen `Punkte` um 1 erhöht.

Außerdem wird der aktuelle Anzeigentext gelöscht und der neue Punktewert angezeigt. Diese beiden Aktionen bündelst du am besten in einer neuen Funktion namens `updateAnzeige()`:

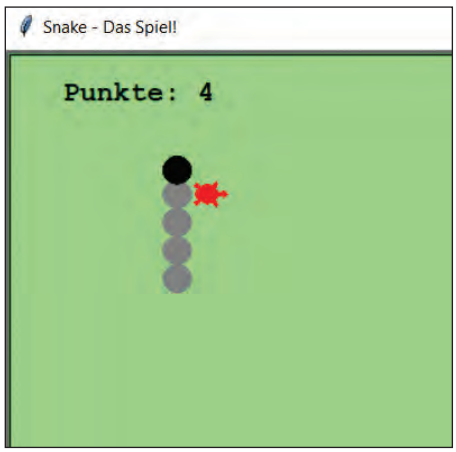
```
def updateAnzeige(): # Aktualisiere die Punktestandsanzeige
   Anzeige.clear()
   Anzeige.write("Punkte: "+str(Punkte),font=("Courier",14,"bold"))
```

Bekommst du das hin? *Zähle die gewonnenen Punkte und zeige sie an.*

Den Code findest du in der Datei Snake3_Futter.py.

Fressen macht groß und stark: Die Schlange wächst

Bisher ist nur der Schlangenkopf herumgelaufen und hat Futter gefressen. Nun soll die Schlange aber noch mit jeder Futterportion ein Segment länger werden. Das macht das Spiel spannender, da man vermeiden muss, dass die Schlange beim Wenden mit ihrem eigenen Körper kollidiert. Wenn das passiert, endet das Spiel.



Schlangensegmente

Wir kümmern uns zunächst noch nicht um das Spielende, sondern um das Wachstum der Schlange. Die einzelnen Körpersegmente, die wir an die Schlange hängen wollen, sind wieder – du ahnst es schon – `Turtle()`-Objekte. Deshalb kannst du sie genau wie den Schlangenkopf bewegen.

Diese Segmente merkst du dir beim Erstellen in einer Liste `Schlangenkoerper`. Damit kannst du sie später zusammen mit dem Kopf der Schlange bewegen. Das letzte Element der Liste ist das zuletzt erstellte Segment.

In der Funktion `neuesSegment()` erstellst und initialisierst du ein neues Schlangensegment:

```
def neuesSegment():
    Segment = Turtle()
    Segment.hideturtle()
    Segment.speed(0)
    Segment.shape("circle")
    Segment.color("grey")
    Segment.penup()
    Segment.setposition(Snake.position()) # Übernimm Position der Schlange
    Segment.showturtle()
    Schlangenkoerper.append(Segment) # Füge Segment der Liste hinzu
```

Bekommst du das hin? *Erstelle immer dann ein neues Segment, wenn die Schlange Futter gefunden hat.*

Die Segmente richtig positionieren

Die Funktion `neuesSegment()` rufst du immer an den Stellen auf, wo die Schlange Futter gefressen hat. Dann zeigt sich ein neues Segment. Aber noch bewegen sich die neuen Segmente mit dem Schlangenkopf nicht mit.

Was muss mit den Segmenten passieren? Wenn sich der Schlangenkopf eine Schrittlänge vorwärts bewegt (in der Funktion `bewegeSchlange()`), dann soll der Schlangenkörper mit allen Segmenten dem Kopf folgen.

Es ist aber sehr mühsam, bei jedem Schritt jedes Segment zu verschieben (vor allem, wenn es viele sind), und es kostet viel Rechenzeit.

Zum Glück ist das auch gar nicht nötig: Denn bei der Bewegung ist an fast allen Stellen, wo vor einem Schritt ein Segment war, hinterher auch wieder eins. Nur das allerletzte Segment ganz hinten am Schwanz der Schlange verschwindet, und eine Lücke ganz vorne direkt hinter dem Kopf entsteht, wenn der Kopf sich nach vorne bewegt.

Am einfachsten ist es daher, wenn wir das letzte Segment aus der Liste löschen und es ganz vorne an erster Stelle wieder in die Liste einfügen. Damit bekommt dieses Segment eine neue Position, nämlich die bisherige Position des Schlangenkopfes.

Dies passiert in der Funktion `updateSchlangenkoerper()`.

```
def updateSchlangenkoerper():
   if len(Schlangenkoerper) > 0:
      # Das letzte Segment übernimmt den ersten Listenplatz:
      letztesSegment = Schlangenkoerper[len(Schlangenkoerper)-1]
      Schlangenkoerper.remove(letztesSegment)
      Schlangenkoerper.insert(0,letztesSegment)
      # und es bekommt die bisherige Schlangenkopfposition:
      letztesSegment.setposition(Snake.position())
```

Die Funktion `updateSchlangenkoerper()` musst du immer dann aufrufen, wenn sich der Schlangenkopf bewegt, also logischerweise in der Funktion `bewegeSchlange()`.

Bekommst du das hin? *Bewege die Schlange und den wachsenden Schlangenkörper. Der Schlangenkörper muss immer dem Kopf folgen.*

Den Code findest du in der Datei Snake4_Wachsen.py.

In der Datei Snake4_Wachsen.py *findest du auch die Alternative für die Funktion* `updateSchlangenkoerper()`*, bei der jedes Mal alle Segmente bewegt werden. Wir haben diese Funktion auskommentiert, das heißt, wir haben in jeder Zeile ein* #*-Zeichen an den Anfang gesetzt. Der Python-Interpreter weiß damit, dass er diese Zeilen ignorieren kann.*

Wenn du die #*-Zeichen löschst und stattdessen die eigentliche Funktion* `updateSchlangenkoerper()` *auskommentierst, kannst du den Code mit der Variante testen. Du wirst merken, dass die Schlange bald sehr langsam wird, weil das Programm so viel rechnen muss.*

Auskommentieren *ist eine Technik, die Profi-Programmierer gerne nutzen. Beispielsweise wenn sie sich nicht sicher sind, welche Variante die bessere ist. Oder wenn sie Code nicht gleich wegwerfen wollen. Wichtig ist, dass du immer dazu schreibst, warum der Code noch im Programm steht und wofür man ihn vielleicht brauchen kann.*

Zu viel Fressen ist ungesund: Die Schlange stirbt

Bisher ist die Schlange herumgelaufen, hat Futter gefressen und Punkte gesammelt. Das könnte natürlich ewig so weitergehen. Ein Spiel macht aber nur Spaß, wenn es nicht zu einfach ist. Daher bauen wir jetzt Schikanen ein, die irgendwann auch dazu führen, dass das Spiel endet.

Bleibe im Spielfeld

Die erste Schikane: Die Schlange darf nicht mit dem Spielfeldrand kollidieren. Tut sie es doch, so ist das Spiel zu Ende.

Wie bekommst du heraus, ob die Schlange mit dem Spielfeldrand kollidiert? Ganz einfach, du fragst bei jedem Schritt ihre Position ab und vergleichst sie mit den Rand-Koordinaten. Das tust du am besten in einer Funktion `wirdRandBeruehrt()`, die `True` ergibt, falls der Rand berührt wurde, und `False`, falls nicht. Mit der Anweisung `return` kannst du `True` oder `False` als

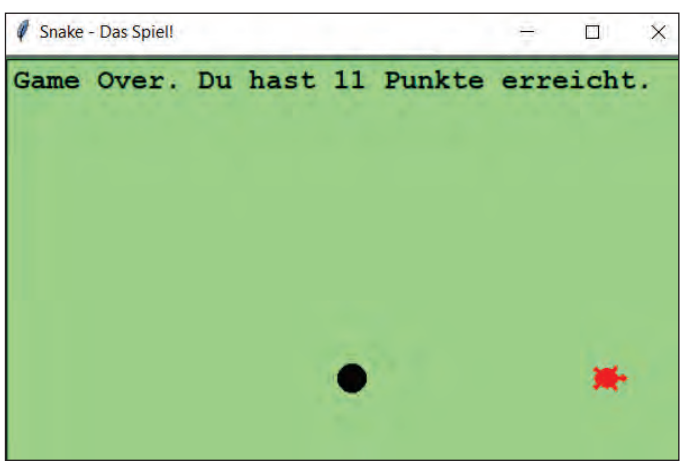
Ergebnis der Funktion ausgeben (der Profi sagt »als Ergebnis der Funktion zurückgeben«):

```
def wirdRandBeruehrt():
   if Snake.xcor() > Breite/2:
      return True
   if Snake.xcor() < -Breite/2:
      return True
   if Snake.ycor() > Hoehe/2:
      return True
   if Snake.ycor() < -Hoehe/2:
      return True
   else:
      return False
```

Auch die Konsequenzen der Spielfeldrandkollision kannst du in eine Funktion packen, die du `Spielende()` nennen könntest. Was passiert alles bei Spielende? Der Schlangenkörper verschwindet, nur der Kopf bleibt übrig. Danach kann das Spiel wieder von vorne beginnen.

```
def Spielende():
   for Segment in Schlangenkoerper:
     Segment.hideturtle()     # mache die Segmente unsichtbar
     Schlangenkoerper[:] = [] # leere die Schlangenkörper-Liste
     Snake.setposition(0,0)   # Setze Schlangenkopf in die Mitte
     Snake.richtung = "keine" # und stoppe die Bewegung
```

Bekommst du das hin? *Ergänze den Code in der* `while-True`*-Schleife so, dass das Spiel endet, wenn der Rand berührt wird. Erweitere die Punkteanzeige, sodass Game Over. Du hast ... Punkte erreicht angezeigt wird.*



Hüte dich vor dem Schlangenkörper

Die zweite Schikane, die das Spiel sehr schnell beenden kann, ist die Regel, dass die Schlange niemals mit dem Kopf in ihren eigenen Schlangenkörper hineinlaufen darf. Diese Regel wird natürlich im Laufe des Spiels immer schwieriger einzuhalten sein, je länger der Schlangenkörper wird.

Ähnlich wie in der Funktion `wirdRandBeruehrt()` soll die Funktion `wirdKoerperBeruehrt()` das Ergebnis `True` zurückgeben, falls der Schlangenkörper mit dem Schlangenkopf `Snake` in Berührung kommt, und `False`, falls nicht. In dieser Funktion kannst du mit einer `for`-Schleife die Segmente in der `Schlangenkoerper`-Liste durchlaufen. Sinnvollerweise prüfen wir die Berührung erst ab dem dritten Segment, da eine kürzere Schlange nicht mit sich selbst kollidieren kann:

```
def wirdKoerperBeruehrt():
   if len(Schlangenkoerper) > 2:
      for index in range(2,len(Schlangenkoerper)-1):
         Segment = Schlangenkoerper[index]
         if Snake.position() == Segment.position():
            return True
      return False
   return False
```

Bekommst du das hin? *Ergänze dein Programm so, dass das Spiel auch dann endet, wenn die Schlange in ihren eigenen Körper hineinläuft. Verwende dazu in der* `while-True`*-Schleife die Funktionen* `wirdKoerperBeruehrt()` *und* `Spielende()`*.*

Es ist für die Spielenden etwas ärgerlich, dass nun jedes Mal das Spiel endet, wenn man den »Rückwärtsgang« einschaltet, also die Schlange dorthin läuft, wo sie gerade herkommt. Beispielsweise läuft die Schlange gerade nach oben und die Spielerin drückt auf ↓. Logisch, dass die Schlange dann in ihren eigenen Körper hineinläuft.

Bekommst du das hin? *Sorge dafür, dass das Spiel nicht endet, wenn der »Rückwärtsgang« eingeschaltet wird. Stattdessen soll die Schlange einfach weiter in die bisherige Richtung laufen.*

Tipp: *Bisher setzten die Funktionen* `gehe_hoch()`*,* `gehe_runter()`*,* `gehe_rechts()` *und* `gehe_links()` *die Richtung der Schlange, ohne zu beachten, wie die Richtung vorher war.*

Hier kannst du nun Bedingungen einfügen, die dafür sorgen, dass die Richtung nur dann geändert wird, wenn nicht die Pfeiltaste für die entgegengesetzte Richtung gedrückt wurde.

Den Code findest du in der Datei Snake5_Spielende.py.

Ideen für Erweiterungen

Du hast es geschafft! Dein *Snake*-Spiel ist fertig.

Vielleicht hast du bereits beim Lesen und Programmieren ein paar neue Ideen bekommen? Ein Profi-Programmierer findet immer was zu verbessern oder zu erweitern. Hier sind einige Vorschläge von uns, die du noch umsetzen könntest, falls du Lust hast.

Mehr Tempo

Um das Spiel noch schwieriger zu machen, kannst du die Schlange abhängig von ihrer Länge schneller werden lassen.

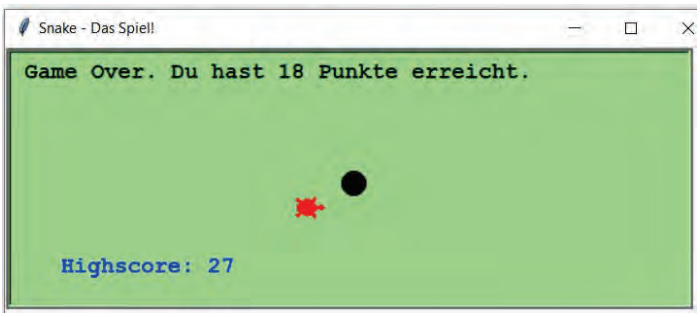
Bekommst du das hin? *Schreibe eine Funktion* `werdeSchneller()`*, die alle 10 Segmente die Geschwindigkeit der Schlange* `Snake.speed()` *um 1 erhöht. Rufe sie auf, nachdem du dem Schlangenkörper ein neues Segment hinzugefügt hast.*

Tipp: *Mit* `int(len(Schlangenkoerper)/10)` *erhältst du die ganze Zahl vor dem Komma, die herauskommt, wenn du die Anzahl der Segmente durch 10 teilst.*

Highscore

Es ist schade, wenn du dir beim Spielen so richtig Mühe gibst, eine hohe Punktzahl zu erreichen – und diese dann mit Beginn des nächsten Spiels gleich wieder verschwindet.

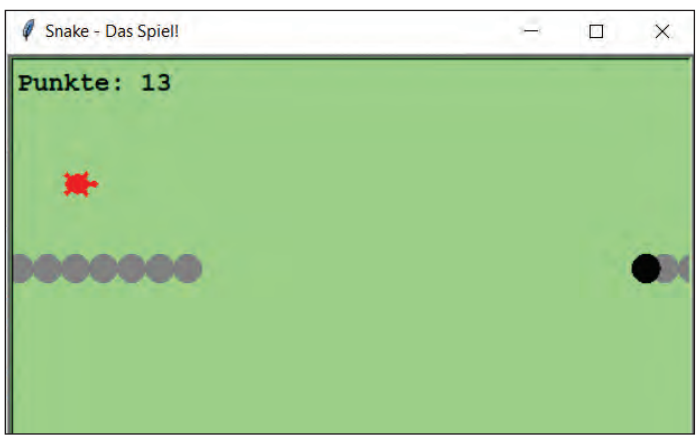
Bekommst du das hin? *Merke dir deinen besten Punktestand über mehrere Spiele hinweg. Zeige diesen Highscore mit einer extra Anzeige an, beispielsweise in der unteren linken Ecke.*



Tipp: *Lege eine Variable* `Highscore` *an, die du bei Spielende mit dem aktuellen Punktestand vergleichst. Die Highscore-Anzeige ist wieder ein* `Turtle()`*-Objekt und wird so ähnlich programmiert wie die Punkteanzeige.*

Durch die Wand gehen

Eine Spielregel ist ja, dass das Spiel endet, sobald die Schlange gegen den Spielfeldrand stößt. Du kannst die Regel aber etwas weniger streng machen, indem du programmierst, dass die Schlange auf der anderen Seite wieder erscheint. Sie verschwindet also links und kommt rechts wieder heraus. Oder sie wandert oben über den Rand und erscheint unten wieder.



Bekommst du das hin? *Lasse die Schlange am Spielfeldrand durch die Wand gehen und am gegenüberliegenden Rand wieder erscheinen.*

Tipp: *Anstatt in der* `while-True`*-Schleife* `Spielende()` *aufzurufen, wenn der Rand berührt wird, rufst du nun eine neue Funktion* `erscheineGegenueber()` *auf:*

```
if wirdRandBeruehrt():
   erscheineGegenueber()
```

Die neue Funktion fragt – wie die Funktion `wirdRandBeruehrt()` *– alle vier Möglichkeiten der Randkollision ab (oben, unten, links, rechts) und ändert dann die jeweilige Snake-Koordinate, entweder mit* `Snake.setx()` *oder mit* `Snake.sety()`.

Damit die Schlange schnell und unsichtbar zum anderen Rand läuft, verstecke sie am Anfang der Funktion mit `Snake.hideturtle()` *und setze ihre Geschwindigkeit hoch mit* `Snake.speed(0)`. *Mache beide Einstellungen am Ende der Funktion wieder rückgängig.*

Den Code mit diesen Erweiterungen findest du in der Datei *Snake6_Erweiterungen.py*. Unter dem Link www.wiley-vch.de/ISBN9783527719952 findest du alle Programme zu dem *Snake*-Spiel.

Das kannst du jetzt

In diesem Kapitel hast du Folgendes gelernt:

- wie du aus einem Screen-Objekt ein Spielfeld machst
- wie du aus Turtle-Objekten Spielfiguren machst
- wie du Spielfiguren mithilfe von Pfeiltasten steuerst
- wie du (Futter-)Objekte erscheinen und verschwinden lässt
- wie du eine Schlange aus immer mehr Segmenten aufbaust
- wie du Kollisionen von Objekten überprüfst und darauf reagierst
- wie du Texte im Spielfenster ausgibst