Inhaltsverzeichnis

1. Vorwort

Dieses Mal ist die Inspiration für ein neues Projekt mein Neffe, der gerne mit mir zusammen mehr über und mit Python lernen möchte.

Wir haben uns darauf verständigt, dass wir in die Spieleentwicklung gehen wollen. Als erstes nehmen wir uns vor, das Spiel „Snake“ zu programmieren.

Bei dem Spiel geht es darum, eine Schlange auf dem Spielfeld hin und her zu bewegen, um Futterbrocken zu essen, die auf dem Spielfeld verteilt sind. Bei jedem Happen verlängert sich die Schlange. Ziel ist es, so viel Essen wie möglich aufzunehmen. Ihr kennt das alle sicher.

Wir werden mit grundlegenden Dingen anfangen, von daher gehe ich davon aus, dass auch Anfänger den Code verstehen werden. Falls dem nicht so sein sollte, meldet Euch gerne bei mir unter papa@papa-programmiert.de, ich bin für Kritik und Anregungen offen und immer zu haben.

Und jetzt viel Spaß!

2. Das Projekt

Wie im Vorwort beschrieben, geht es in diesem Projekt um ein Spiel, in dem sich eine Schlange in einem Fenster bewegt, um Futter aufzunehmen und so zu wachsen.

Das Bild dazu sieht dann etwa so aus:

Nicht sehr spektakulär, der Phantasie sind allerdings keine Grenzen gesetzt, wem die Farbgestaltung nicht zusagt, kann sich nachher noch richtig austoben, wir werden uns das im Detail anschauen.

Noch ein Hinweis zur Aufteilung der Kapitel.

Manche Kapitel sind länger, andere sind kürzer. Das ist dem Umfang des jeweiligen Themas geschuldet. Am Ende eines jeden Kapitels steht also entweder eine neue Version des Codes bereit, oder ein Thema ohne neuen Code ist beschrieben worden.

Ladet Euch den neuen Stand am Ende des Kapitels herunter, und führt ihn im Editor Eurer Wahl aus.

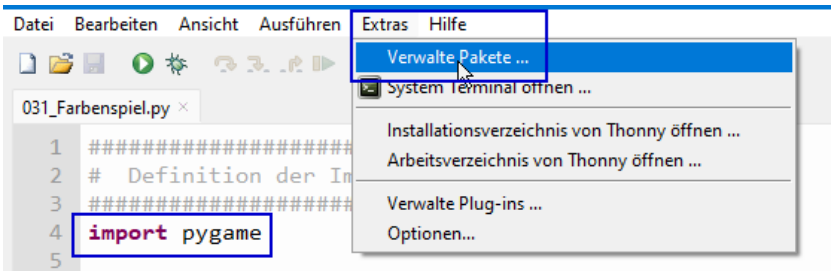
3. Vorarbeiten

Zum vorherigen Projekt gibt es die Vorarbeiten betreffend keine große Veränderung.

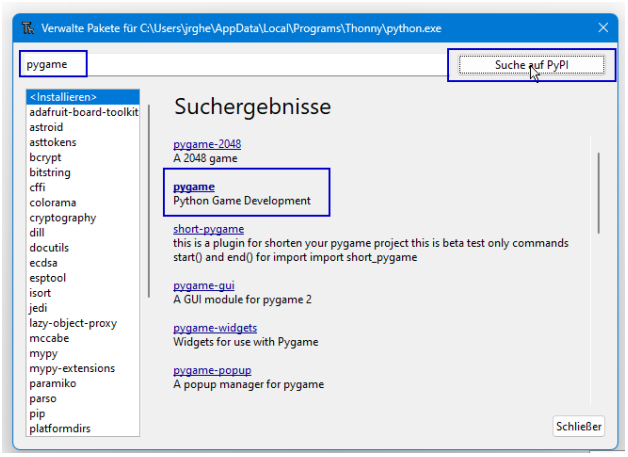
Die Entwicklung habe ich damals in der Thonny IDE gemacht, die werde ich auch hier wieder nutzen. Den Download gibt es auf https://thonny.org/ - zur Entstehung dieses Projektes ist das ist die Version 4.0.1, die habe ich mir jetzt installiert. Diese Version läuft mit Python 3.10.6.

Für die Spieleentwicklung stehen diverse Bibliotheken zur Verfügung, wir nutzen `pygame`. Diese Bibliothek müssen wir zunächst installieren.

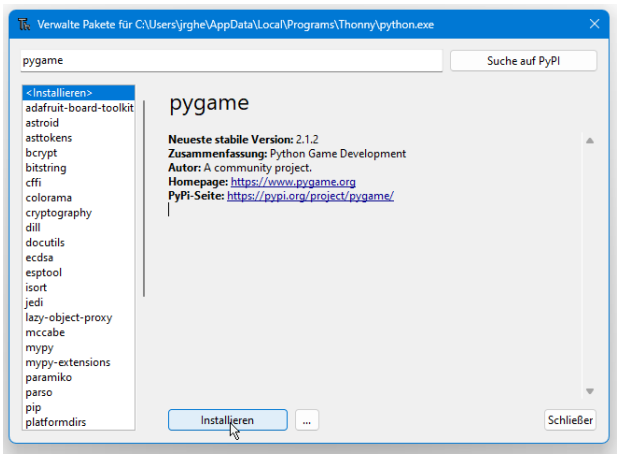
Über den Menüpunkt „Extras/Verwaltete Pakete…“:



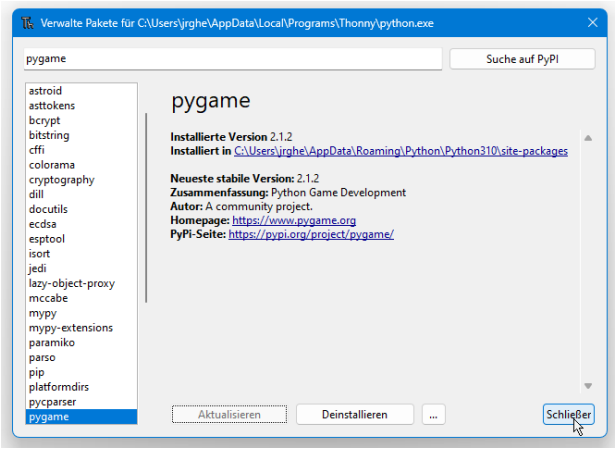
gelangen wir in den Paket-Assistenten:



In der Suchleiste „pygame“ eigeben, dann „Suchen auf PyPI“ klicken und das Paket pygame anklicken.



Im folgenden Fenster „Installieren“ klicken.



Mit „Schließen“ den Assistenten verlassen.

Das war es dann auch schon mit den Vorbereitungen, genug der Vorarbeiten, los geht’s!

4. Projekt „Snake"

Wie auch in den anderen Projekten schon praktiziert, machen wir uns zuerst Gedanken über den Funktionsumfang, also die Frage nach dem ‚**Was**'. Das strukturiert das Projekt und wir haben eine klarere Vorstellung, als wenn wir direkt „draufloshacken" würden.

Dann gehen wir das ‚**Wie**' an. Fest steht ja schon, dass wir das mit Python lösen wollen, wir schauen aber nochmal auf die benötigten Pakete.

Im dritten Teil dann kommt die Programmierung dran.

Soweit so gut, gehen wir es an.

4.1. Der Funktionsumfang

Da wir uns in einem Spiel befinden, ist der Funktionsumfang gleichzusetzen mit den Spielregeln.

Das Programm soll dem Spieler die Möglichkeit geben, einen Punkt auf dem Spielefenster mittels der Hoch-/Runter-/Links-/Rechts-Tasten im Fenster zu bewegen.

Im Fenster wird jeweils ein Block als „Futter" platziert, diesen Block muss die Schlange erreichen, dann verlängert sich der Schwanz der Schlange um die Länge des Blocks.

Gerät der Spieler mit der Schlange an den oberen, unteren, linken oder rechten Rand, ist das aktuelle Spiel verloren.

Der Spieler soll die Auswahl haben, ob er das ganze Spiel verlassen möchte, oder ein neues Spiel starten will.

In einer weiteren Ausbaustufe kann die Geschwindigkeit der Schlange erhöht werden, der Highscore kann gespeichert werden oder mehrere Spieler sollen abwechselnd spielen können. Das ist aber nicht mehr Teil dieses Projektes.

4.2. Die Planung der Umsetzung

Wir werden in mehreren Schritten vorgehen.

Zunächst legen wir ein Grundgerüst fest, das uns als Vorbild für weitere Programme dienen kann.

Dann schauen wir auf die grundsätzliche Idee hinter der Bibliothek `pygame`.

Wenn wir uns damit vertraut gemacht haben, lassen wir die Schlange entstehen und kümmern uns um die Bewegung im Fenster.

Im Anschluss legen wir das Futter aus und lassen die Schlange länger werden.

Zu guter Letzt geben wir noch den Spielstand aus und ermöglichen eine Wiederholung oder den Ausstieg.

Die weiteren Ausbaustufen werden wir nicht angehen, vielleicht machen wir dazu ja noch ein eigenes Folgeprojekt, lassen wir uns überraschen.

Jetzt geht es erst einmal los mit dem Coden.

4.3. Das Coden

Bitte gestattet mir vor dem ersten Code-Beispiel noch ein paar allgemeine Gedanken zum Thema „Programmieren in Python“.

Den Anfang macht das **Grundgerüst**.

Warum ein Grundgerüst? Aus meiner Sicht strukturiert das den Code enorm und man ist auch nach Wochen noch in der Lage, sich in seinem eigenen Code zurecht zu finden. Ich würde gerne hier schon auf ein paar Besonderheiten eingehen, die ich für gut befunden habe. Das muss natürlich niemand genauso machen, ich will da niemanden bevormunden.

Ich liebe die **Funktionen** in Python. Sie helfen mir, Dinge und Abläufe zu strukturieren. Ich habe mir angewöhnt, die Variablen außerhalb der Funktionen zu deklarieren, dann stehen alle mehr oder weniger an einem Platz. Das impliziert allerdings, dass jede in einer Funktion genutzten Variable zuerst mit global gekennzeichnet wird. Andernfalls ist es eine lokale Variable, die ich außerhalb der Funktion nicht mehr verwenden kann.

Damit erspare ich mir sehr häufig, dass Funktionen Parameter übergeben bekommen, oder mittels `return` Parameter zurückgeben. Die bräuchten innerhalb der Funktion andere Namen als außerhalb, was mich immer wieder selbst verwirrt. Ich zeige Euch gleich, was ich meine.

Natürlich definiere ich auch in den Funktionen lokale Variablen, aber die bleiben dann tatsächlich lokal und werden nicht zurückgegeben.

Die Funktionen lasse ich alle mit einem `print()` starten, sodass ich die Struktur in der Ausgabe kontrollieren kann. In der endgültigen Version muss das dann natürlich raus, aber bis dahin haben wir noch viel Zeit.

Ein weiteres Thema sind **Kommentare**. Ich füge immer dann gerne Kommentarblöcke ein, wenn es einen neuen Abschnitt gibt. Also alle (globalen) Variablen an einem Ort, alle Funktionen schön übersichtlich und auch den eigentlichen Programmstart schön getrennt vom Rest.

Und nun noch ein Wort zur **Darstellung des Codes**. In den letzten Projekten hat sich bewährt, neu eingeführten Code grün einzufärben, dann ist sofort ersichtlich, was neu ist. Das werde ich auch hier wieder so tun.

Das waren meine einleitenden Worte zum Thema **Coden**, jetzt legen wir los!

4.3.1. Das Grundgerüst

Für die Programmierung mit der Bibliothek `pygame` habe ich mir folgende Grundstruktur überlegt:

```
1  ###################################################
2  #  Definition der Importe
3  ###################################################
4  import pygame
5
6  ###################################################
7  #  Definition der Konstanten und Variablen
8  ###################################################
9
10 #  Werte für das Spielefenster
11 BREITE = 800
12 HOEHE = 400
13
14 # Spielsteuerung
15 spielen = True
16
17 ###################################################
18 #  Definition der Funktionen
19 ###################################################
20
21 def fenster_definieren():
22     print('in fenster_definieren)
23     pygame.init()
24     f = pygame.display.set_mode((BREITE,HOEHE))
25     pygame.display.set_caption('Snake - das Schlangenspiel')
26     return f
27
28
29 def spielschleife_aufrufen():
30     print('in spielschleife_aufrufen')
31     global spielen
32     while spielen:
33         print('in der Schleife selbst')
34         spielen = False
35
36
37 def spiel_verlassen():
38     print('in spiel_verlassen')
39     pygame.quit()
40     quit()
41
42
43 ###################################################
44 #  Programmstart
45 ###################################################
46
47 print('Programmstart!')
48
49 fenster = fenster_definieren()
50
51 spielschleife_aufrufen()
52
53 spiel_verlassen()
54
```

Sieht aufgeräumt aus, oder?

Gehen wir es kurz durch.

Zur besseren Lesbarkeit habe ich wie oben beschrieben **4 Kommentarblöcke** eingefügt, die `Definition der Importe`, `Definition der Variablen`, `Definitionen der Funktionen` und den `Programmstart`.

Warum ist der Programmstart jetzt unten? Das Programm fängt doch schon mit dem ersten `import` an?

Berechtigte Frage. Für mich ist alles, was vor Zeile 47 steht, die Deklaration des Programms. Also die grundsätzlichen Vorbedingungen, die gegeben sein müssen, damit das Programm läuft.

Also von oben nach unten, zuerst alle Importe. Im weiteren Verlauf brauchen wir noch eine weitere (`random`).

Der nächste Block sind die Konstanten und Variablen. Der Unterschied ist klar, oder? Konstanten verändern sich nicht während der Laufzeit des Programms, Variablen dagegen schon. Um das zu kennzeichnen, werden Konstanten durchgängig GROSS geschrieben, Variablen durchgängig klein.

Die beiden Konstanten `BREITE` und `HOEHE` sind die Pixel-Angaben für das Spielfenster, wir brauchen sie in Zeile 24.

Die Variable `spielen` ist vom Typ `boolean` (also „wahr“ oder „unwahr“). Sie ist die zentrale Abbruchbedingung unserer Schleife um das ganze Spiel. Zu sehen ist das in Zeile 32.

Die Funktionen überspringe ich kurz, ich mache weiter mit Zeile 47. Hier beginnt für mich das eigentliche Programm. Der Start wird in Zeile 46 mit der `print()`-Funktion ausgegeben, das kennen wir ja schon.

Das eigentliche Programm besteht als aus den 3 hintereinander laufenden Funktionen `fenster_definieren()`, `spielschleife_aufrufen()` und `spiel_verlassen()`.

Was machen die einzelnen Funktionen nun?

In `fenster_definieren()` wird die Bibliothek `pygame` intitialisert (Zeile 23) und das Fenster wird erzeugt. Hier ist das, was ich in Kapitel 4.2 angesprochen habe. In der Funktion wird die lokale Variable `f` benötigt, um das Fenster zu instanziieren. Die lokale Variable `f` wird zurückgegeben und in Zeile 48 in der globalen (weil außerhalb der Funktion benötigten) Variable `fenster` gespeichert.

Die Funktion `spielschleife_aufrufen()` beherbergt unsere zentrale Spielsteuerung, nämlich die Schleife `while spielen:`. Solange die Variable `spielen` wahr ist (also `True`) wird die Schleife nicht verlassen. Und hier sehen wir die vorangestellte Deklaration `global spielen`, das sagt uns, dass wir im Verlauf der Funktion die *globale* Variable `spielen` (die in Zeile 15, also *außerhalb* der Funktion definiert wurde) ansprechen und ändern wollen.

In unserem Gerüst setzen wir `spielen` sofort nach der Ausgabe von `'in der Schleife selbst'` auf unwahr (also `False`), das beendet die Schleife nach nur einem Durchgang.

Weiter geht es dann mit der Funktion `spiel_verlassen()` in der das Fenster und auch das Programm selbst beendet werden.

Soviel zum Grundgerüst.

Wenn Ihr das Programm laufen lasst, passiert so gut wie nichts, einmal kurz flackert ein Fenster auf, das geht aber gleich wieder zu. In der Ausgabe der Kommandozeile von Thonny sollten aber sämtliche `print()`-Anweisungen erscheinen:

```
Kommandozeile
>>> %Run 4_3_1_Grundgeruest.py
  pygame 2.1.2 (SDL 2.0.18, Python 3.10.6)
  Hello from the pygame community. https://www.pygame.org/contribute.html
  Programmstart!
  in fenster_definieren
  in spielschleife_aufrufen
  in der Schleife selbst
  in spiel_verlassen

  Process ended with exit code 0.
```

Damit ist der Grundstock gelegt, machen wir also weiter.

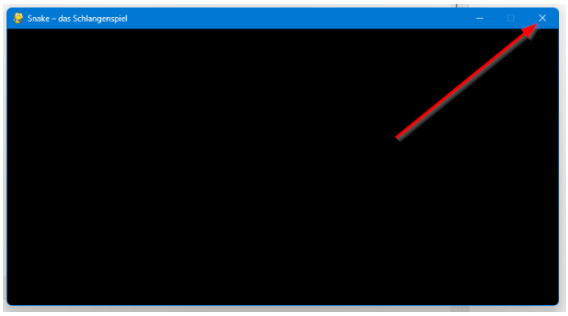
4.3.2. Das Fenster

Um die Arbeitsweise von `pygame` besser zu verstehen, erweitern wir zunächst die Spielschleife. Nachfolgend grün die Änderungen im Code:

```
29 def spielschleife_aufrufen():
30     print('in spielschleife_aufrufen')
31     global spielen
32     while spielen:
33         for event in pygame.event.get():
34             # Ausgeben aller Aktivitaeten, die innerhalb des Fensters stattfinden
35             print(event)
36             # wenn das x im Fenster gedrueckt wird, soll das Fenster zugehen
37             if event.type == pygame.QUIT:
38                 spielen = False
39
```

Alle Aktivitäten, die in einem mit `pygame` erzeugten Fenster passieren, können über die Funktion `pygame.event.get()` abgefragt werden. Innerhalb unserer `while`-Schleife fragen wir permanent die im Fenster stattfindenden Aktivitäten ab und geben diese mit der `print()`-Funktion in Zeile 35 aus.

In Zeile 37 fangen wir ein spezielles `event` ab, nämlich der Klick auf das Kreuz im Fenster:




```
Kommandozeile
  <Event(1024-MouseMotion {'pos': (742, 4), 're
  <Event(1024-MouseMotion {'pos': (742, 3), 're
  <Event(1024-MouseMotion {'pos': (744, 1), 're
  <Event(1024-MouseMotion {'pos': (745, 1), 're
  <Event(1024-MouseMotion {'pos': (747, 0), 're
  <Event(1024-MouseMotion {'pos': (747, 0), 're
  <Event(32768-ActiveEvent {'gain': 0, 'state':
  <Event(32784-WindowLeave {'window': None})>
  <Event(32787-WindowClose {'window': None})>
  <Event(256-Quit {})>
  in spiel_verlassen
```

Auch die Tastatur wird überwacht. Nachfolgendes Beispiel aus der Kommandozeile:

```
<Event(1024-MouseMotion {'pos': (31, 9), 'rel': (0, -3), 'buttons': (0, 0, 0), 'touch': False, 'window': None})>
<Event(1024-MouseMotion {'pos': (29, 5), 'rel': (-2, -4), 'buttons': (0, 0, 0), 'touch': False, 'window': None})>
<Event(1024-MouseMotion {'pos': (28, 2), 'rel': (-1, -3), 'buttons': (0, 0, 0), 'touch': False, 'window': None})>
<Event(1024-MouseMotion {'pos': (25, 0), 'rel': (-3, -8), 'buttons': (0, 0, 0), 'touch': False, 'window': None})>
<Event(32768-ActiveEvent {'gain': 0, 'state': 0})>
<Event(32784-WindowLeave {'window': None})>
<Event(768-KeyDown {'unicode': 'q', 'key': 113, 'mod': 4096, 'scancode': 20, 'window': None})>
<Event(771-TextInput {'text': 'q', 'window': None})>
<Event(769-KeyUp {'unicode': 'q', 'key': 113, 'mod': 4096, 'scancode': 20, 'window': None})>
<Event(768-KeyDown {'unicode': 'w', 'key': 119, 'mod': 4096, 'scancode': 26, 'window': None})>
<Event(771-TextInput {'text': 'w', 'window': None})>
<Event(769-KeyUp {'unicode': 'w', 'key': 119, 'mod': 4096, 'scancode': 26, 'window': None})>
<Event(768-KeyDown {'unicode': 'e', 'key': 101, 'mod': 4096, 'scancode': 8, 'window': None})>
<Event(771-TextInput {'text': 'e', 'window': None})>
<Event(769-KeyUp {'unicode': 'e', 'key': 101, 'mod': 4096, 'scancode': 8, 'window': None})>
<Event(32787-WindowClose {'window': None})>
<Event(256-Quit {})>
in spiel_verlassen
```

Die ersten 4 Einträge zeigen meine Mausbewegung, dann hat der Mauszeiger das Fenster verlassen und der Buchstabe ‚q‘ wurde eingegeben.

Das führt zu 3 `events`, einmal dem `KeyDown`, also dem Herunterdrücken der Taste ‚q‘, dem `event TextInput` mit ‚q‘ und dem `event KeyUp` für das Loslassen der Taste ‚q‘.

Das `envent Quit` nutzen wir, um die Abbruchbedingung für die Schleife zu setzen (Zeilen 37 – 38).

Probiert es aus, macht Euch vertraut mit den `events`.

4.3.3. Der Schlangenkopf

Als nächstes gehen wir den Schlangenkopf an.

Der Einfachheit halber, werden wir als Kopf nur ein Quadrat mit einer Kantenlänge von 10 Pixeln definieren. Der Hintergrund ist schwarz, also nehmen wir das erste Quadrat in rot auf.

In `pygame` werden die Zahlen mittels RGB-Werten angegeben. Wer es noch nicht kennt, das ‚R‘ steht für „red“, das ‚G‘ für „green“ und das ‚B‘ für „blue“. Angegeben werden jeweils Werte zwischen 0 und 255. ‚0‘ bedeutet dabei, nichts von der Farbe, 255 das volle Programm der Farbe.

Die gängigsten Farben habe ich nachfolgend schon aufgenommen, obwohl wie aktuell nur rot verwenden:

```
 6 ##################################################
 7 #  Definition der Konstanten und Variablen
 8 ##################################################
 9 …
17 # Farben RGB (Red, Green, Blue), jeweils 0 - 255
18 ROT = (255,0,0)
19 GRUEN = (0,255,0)
20 BLAU = (0,0,255)
21 WEISS = (255, 255, 255)
22 SCHWARZ = (0, 0, 0)
23 …
```

Die Erweiterung in der Schleife sieht wie folgt aus:

```
35 …
36 def spielschleife_aufrufen():
37     print('in spielschleife_aufrufen')
38     global spielen
39     while spielen:
40         for event in pygame.event.get():
41             # Ausgeben aller Aktivitäten, die innerhalb des Fensters stattfinden
42             print(event)
43             # wenn das x im Fenster gedrückt wird, soll das Fenster zugehen
44             if event.type == pygame.QUIT:
45                 spielen = False
46
47         # den Kopf der Schlange symbolisiert ein Quadrat
48         pygame.draw.rect(fenster,ROT,[400,200,10,10])
49         pygame.display.update()
50  …
```

In diesem Anschnitt sind nur die Zeilen 47 bis 49 neu eingefügt.

In Zeile 49 ergeht der Befehl an `pygame`, ein Rechteck zu zeichnen. Mitzugeben sind der Name des Fensters (`fenster`), die Farbe (`ROT`) und die Positionsangaben.

Die `400` steht dabei für den Wert auf der x-Achse, beginnend bei 0 am linken Rand. Da die Breite des Fensters mit 800 Pixeln festgelegt ist, ist 400 genau die Hälfte.

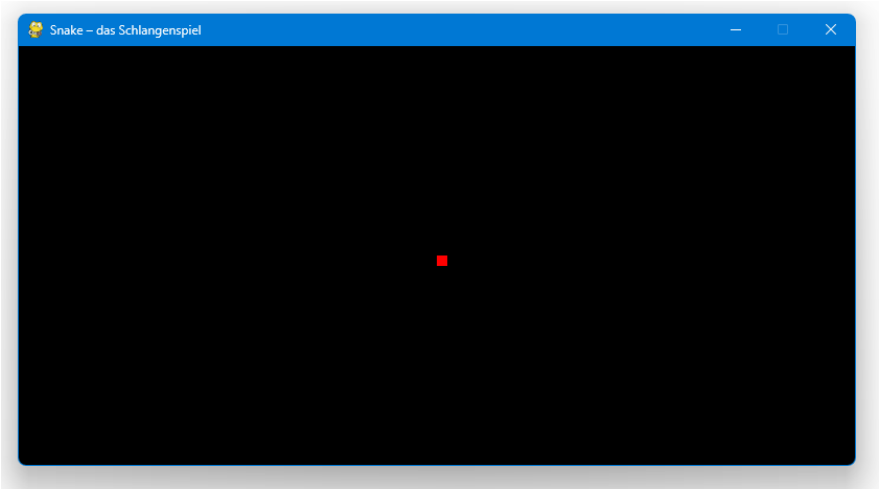
Die `200` steht analog damit für den Wert auf der y-Achse. Diese beginnt am oberen Rand mit 0. Unsere Höhe ist 400, also auch hier die Hälfte.

Die erste `10` steht für 10 Pixel Kantenlänge der x-Achse, die zweite `10` für 10 Pixel auf der y-Achse.

Streng genommen ist das Quadrat damit nicht genau auf der Hälfte des Fensters positioniert. Nur der obere linke Punkt ist genau auf der Hälfte. Damit auch das Quadrat genau in der Mitte ist, müssten wir jeweils die Hälfte der Seite von der x- und der y-Position abziehen.

Ich weiß, niemand mag Klugscheißer…

Ganz wichtig ist der Eintrag in Zeile 50. Ohne den Befehl `pygame.display.update()` wird keine der gemachten Änderungen im Fenster angezeigt! Was sehen wir bei der Ausführung des Programms?



Ein rotes Rechteck auf schwarzem Grund. Cool.

4.3.4. Die Bewegung der Schlange

Jetzt machen wir wieder einen etwas größeren Schritt.

Wir erinnern uns, die Schlange soll sich ja frei im Fenster bewegen können. Dazu nutzen wir die Tastatur um Befehle entgegen zu nehmen.

Ich habe mich für die Richtungspfeile () entschieden, Ihr könnt aber auch jede andere Tastenkombination nutzen.

Die neuen (globalen!) Variablen lauten wie folgt:

```
23 …
24 # Positionen der oberen linken Ecke des Kopfes
25 snake_x = 400
26 snake_y = 200
27
28 # Groesse des Quadrats
29 quadrat_seite = 10
30
31 # Bewegungen auf der x- und der y-Achse
32 snake_bewegung_x = 0
33 snake_bewegung_y = 0
34
35 # Geschwindigkeiten
36 zeit = pygame.time.Clock()
37 snake_geschwindigkeit = 20
38 …
```

Mit `snake_x`, `snake_y` und `quadrat_seite` ersetzen wir gleich die Werte aus dem Zeichnen des Rechtecks.

Die Bewegung findet durch Veränderung der Werte auf der x- oder der y-Achse statt. Wie genau das funktioniert, schauen wir uns gleich an. Wir brauchen aber 2 Variablen, in denen die die neuen Werte speichern. Das sind `snake_bewegung_x` und `snake_bewegung_y`.

Die beiden letzten Variablen brauchen wir für die Wiederholfrequenz in der der Bildschirm aktualisiert wird. Dazu dient die Funktion `pygame.time()`, sie erwartet einen Wert in Millisekunden, ich habe hier mal 20 angegeben und das gleich in der Variable `snake_geschwindigkeit` verpackt.

Ändert das mal und schaut, was dann passiert. Spielt auch mal mit den anderen Variablen diverse Konstellationen durch.

Die Spielschleife ändert sich natürlich auch:

```
50 …
51 def spielschleife_aufrufen():
52     print('in spielschleife_aufrufen')
53     global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y
54     while spielen:
55         for event in pygame.event.get():
56             # Ausgeben aller Aktivitäten, die innerhalb des Fensters stattfinden
57             # print(event)
58             # wenn das x im Fenster gedrückt wird, soll das Fenster zugehen
59             if event.type == pygame.QUIT:
60                 spielen = False
61
62             # die Tastatur abfragen
63             if event.type == pygame.KEYDOWN:
64                 if event.key == pygame.K_RIGHT:
65                     snake_bewegung_x = quadrat_seite
66                     snake_bewegung_y = 0
67                 elif event.key == pygame.K_LEFT:
68                     snake_bewegung_x = quadrat_seite * -1
69                     snake_bewegung_y = 0
70                 elif event.key == pygame.K_UP:
71                     snake_bewegung_x = 0
72                     snake_bewegung_y = quadrat_seite * -1
73                 elif event.key == pygame.K_DOWN:
74                     snake_bewegung_x = 0
75                     snake_bewegung_y = quadrat_seite
76
77         snake_x += snake_bewegung_x
78         snake_y += snake_bewegung_y
79
80         pygame.draw.rect(fenster, ROT, [snake_x, snake_y, quadrat_seite, quadrat_seite])
81         pygame.display.update()
82         zeit.tick(snake_geschwindigkeit)
83 …
```

Sieht viel aus, ist aber nicht kompliziert, wenn man weiß, wie das Zusammenspiel ist.

Da ich mich für globale Variablen entschieden habe, brauchen wir in Zeile 53 die Erweiterung um die neuen Variablen.

In Zeile 57 ist die Ausgabe von `event` auskommentiert.

Zwischen Zeile 63 und 75 passieren 2 Dinge.

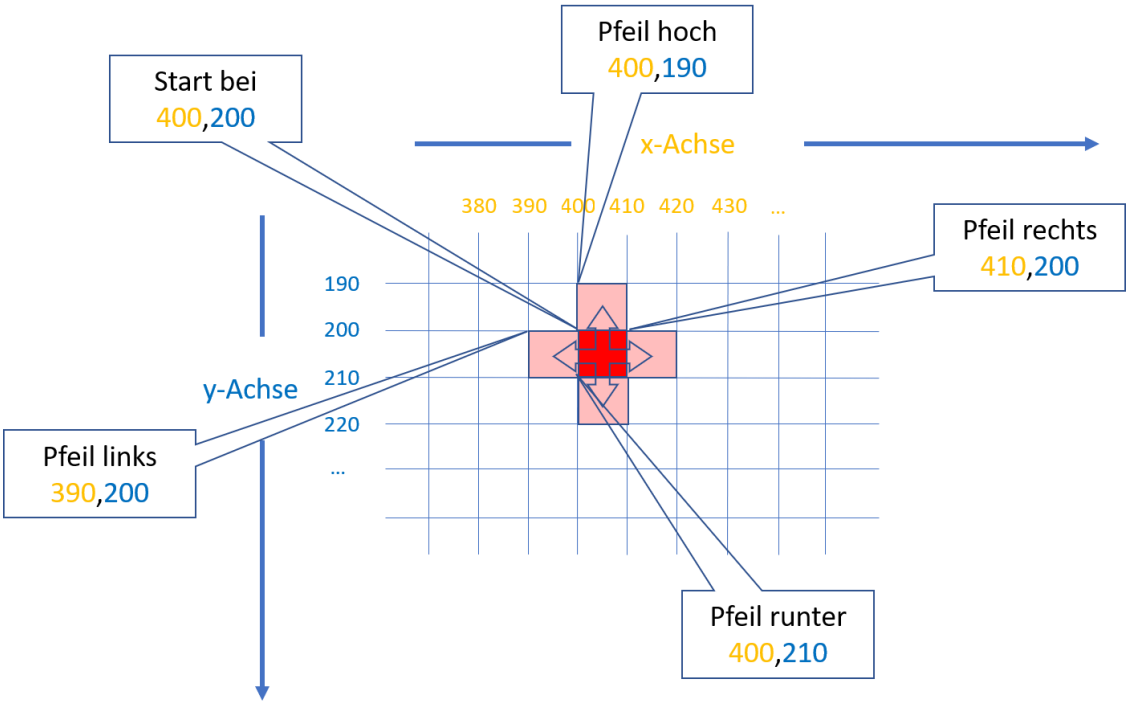
Zuerst „fangen“ wir in Zeile 63 das `event` vom Typ ‚eine-Taste-auf-der-Tastatur-wurde-gedrückt‘ ab (`pygame.KEYDOWN`).

In den Zeilen 64, 67, 70 und 73 kümmern wir uns nur um die 4 Pfeil-Tasten, alle anderen Tasten werden nicht berücksichtigt. Falls Ihr andere Tasten nutze wollt, schaut im Internet in der offiziellen Dokumentation von `pygame` nach, die Adresse ist https://www.pygame.org/docs/ref/key.html.

Entscheidet Ihr Euch zum Beispiel für die bei Spielen ebenfalls gängige ‚WASD‘-Kombination, wäre das Äquivalent zu `pygame.K_UP` die Abfrage auf `pygame.K_w` zu machen.

Was aber muss passieren, damit wir das Quadrat in Bewegung bekommen?

Ganz einfach, die Position der oberen linken Ecke des Quadrats muss geändert werden, und zwar genau um die Seitenlänge des Quadrats. Nachfolgendes Bildchen soll den ersten Schritt nach Start des Fensters symbolisieren. Los geht es bei uns ja an der Position x = 400 und y = 200:



- Erfolgt die Bewegung nach **rechts** muss zum aktuellen Wert der x-Achse die Seitenlänge des Quadrats addiert werden (also aus 400 wird 410). Der Wert für die y-Achse bleibt bei 200.
- Erfolgt die Bewegung nach **links** muss vom aktuellen Wert der x-Achse die Seitenlänge des Quadrats subtrahiert werden (also aus 400 wird 390). Der Wert für die y-Achse bleibt bei 200.
- Erfolgt die Bewegung nach **oben** muss vom aktuellen Wert der y-Achse die Seitenlänge des Quadrats subtrahiert werden (also aus 200 wird 190). Der Wert für die x-Achse bleibt unverändert bei 400.
- Erfolgt die Bewegung nach **unten** muss zum aktuellen Wert der y-Achse die Seitenlänge des Quadrats addiert werden (also aus 200 wird 210) Der Wert für die x-Achse bleibt bei 400.

Nun zur zweiten Aktion, wir weisen die neuen Werte den beiden Variablen `snake_bewegung_x` und `snake_bewegung_y` zu. Entweder ist es dann der Wert von `quadrat_seite` selbst (in unserem Fall also ‚10‘), oder der Kehrwert durch die Multiplikation mit -1 (also ‚-10‘).

In den Zeilen 77 und 78 erfolgt dann die Anpassung der beiden Variablen `snake_x` und `snake_y`.

Zeile 80 hatten wir schon, allerdings sind jetzt die fixen Werte 400, 200, 10 und 10 durch die Variablen ersetzt worden. An der grundsätzlichen Verarbeitung hat sich also nichts geändert.

Zeile 82 ist neu, hier kommt jetzt die erwähnte zeitliche Komponente dazu. Wir befinden uns ja immer noch in der `while`-Schleife aus Zeile 54. Erst, wenn oben rechts das X geklickt wird, verlassen wir die Schleife durch setzen der Variablen `spielen` auf `False`.

Der Aufruf von `zeit.tick()` bewirkt, dass so etwas wie eine ‚Pause' in der Verarbeitung der Schleife eingebaut wird. Wie groß diese Pause ist, drückt die Zahl aus, die in der Klammer mitgegeben wird. Sie repräsentiert den Bruchteil einer Sekunde.

Da wir dafür die Variable `snake_geschwindigkeit` definiert und diese mit ‚20' vorbelegt haben (Zeile 37), sorgt `zeit.tick(snake_geschwindigkeit)` dafür, dass die `while`-Schleife 20-mal pro Sekunde durchlaufen wird.

Und solange wir keine andere Pfeiltaste drücken, wird die Richtung beibehalten.

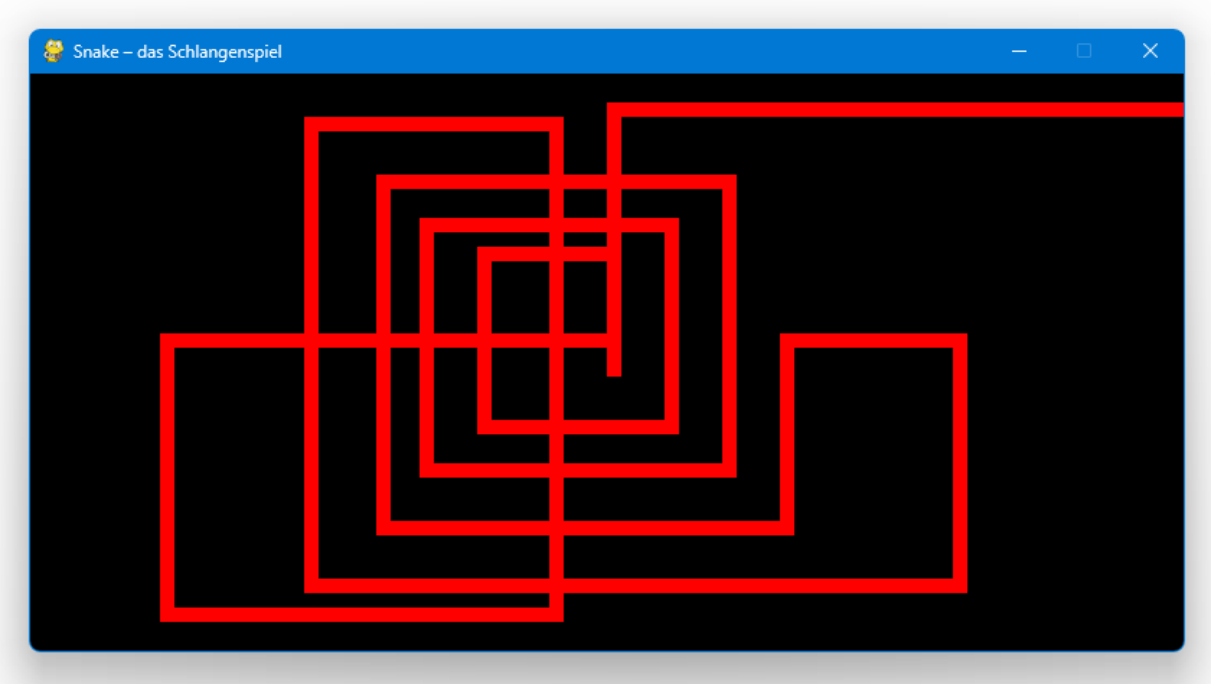
Schauen wir es uns an. Da ja doch einige Änderungen gemacht wurden, hier noch einmal der aktuelle Code in Gänze:

```
 1 ##################################################
 2 #  Definition der Importe
 3 ##################################################
 4 import pygame
 5
 6 ##################################################
 7 #  Definition der Konstanten und Variablen
 8 ##################################################
 9
10 #  Werte für das Spielefenster
11 BREITE = 800
12 HOEHE = 400
13
14 # Spielsteuerung
15 spielen = True
16
17 # Farben RGB (Red, Green, Blue), jeweils 0 - 255
18 ROT = (255,0,0)
19 GRUEN = (0,255,0)
20 BLAU = (0,0,255)
21 WEISS = (255, 255, 255)
22 SCHWARZ = (0, 0, 0)
23
24 # Positionen der oberen linken Ecke des Kopfes
25 snake_x = 400
26 snake_y = 200
27
28 # Bewegungen auf der x- und der y-Achse
29 snake_bewegung_x = 0
30 snake_bewegung_y = 0
31
32 # Geschwindigkeiten
33 zeit = pygame.time.Clock()
34 snake_geschwindigkeit = 20
35
36 # Groesse des Quadrats
37 quadrat_seite = 10
38
39 ##################################################
40 #  Definition der Funktionen
41 ##################################################
42
43 def fenster_definieren():
44     print('in fenster_definieren')
```

```
45      pygame.init()
46      f = pygame.display.set_mode((BREITE,HOEHE))
47      pygame.display.set_caption('Snake – das Schlangenspiel')
48      return f
49
50
51  def spielschleife_aufrufen():
52      print('in spielschleife_aufrufen')
53      global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y
54      while spielen:
55          for event in pygame.event.get():
56              # Ausgeben aller Aktivitäten, die innerhalb des Fensters stattfinden
57              # print(event)
58              # wenn das x im Fenster gedrückt wird, soll das Fenster zugehen
59              if event.type == pygame.QUIT:
60                  spielen = False
61
62              # die Tastatur abfragen
63              if event.type == pygame.KEYDOWN:
64                  if event.key == pygame.K_RIGHT:
65                      snake_bewegung_x = quadrat_seite
66                      snake_bewegung_y = 0
67                  elif event.key == pygame.K_LEFT:
68                      snake_bewegung_x = quadrat_seite * -1
69                      snake_bewegung_y = 0
70                  elif event.key == pygame.K_UP:
71                      snake_bewegung_x = 0
72                      snake_bewegung_y = quadrat_seite * -1
73                  elif event.key == pygame.K_DOWN:
74                      snake_bewegung_x = 0
75                      snake_bewegung_y = quadrat_seite
76
77          snake_x += snake_bewegung_x
78          snake_y += snake_bewegung_y
79
80          pygame.draw.rect(fenster, ROT, [snake_x, snake_y, quadrat_seite, quadrat_seite])
81          pygame.display.update()
82          zeit.tick(snake_geschwindigkeit)
83
84  def spiel_verlassen():
85      print('in spiel_verlassen')
86      pygame.quit()
87      quit()
88
89
90  ###################################################
91  #  Programmstart
92  ###################################################
93
94  print('Programmstart!')
95
96  fenster = fenster_definieren()
97
98  spielschleife_aufrufen()
99
100 spiel_verlassen()
101
```

Was sehen wir jetzt?

Bei mir sieht das nach einigem Herumspielen so aus:



Schon ganz schick, es bewegt sich zumindest etwas.

Wie man aber sieht, habe ich keine *Schlange bewegt* sondern es wurde *eine Spur gezogen*. Auf der rechten Seite bin ich aus dem Fenster gelaufen, das müssen wir ändern!

Und darum kümmern wir uns im nächsten Kapitel.

4.3.5. Kleine Verbesserungen

Das letzte Kapitel hatte es in sich, Ihr habt das aber sehr gut gemeistert! Als „kleine Zwischenmahlzeit“ gewissermaßen hier 2 kleine Verbesserungen.

Der Grund dafür, dass wir eine Spur ziehen und nicht die Schlange bewegen, liegt darin, dass wir hinter der Schlange noch nicht aufräumen.

Das Fenster wird standardmäßig mit schwarzem Hintergrund ausgeliefert. Das geschieht aber nur einmalig bei Start des Programms. Deshalb müssen wir in die Schleife vor dem Aufruf von `pygame.display.update()` noch das Fenster füllen. Der Befehl dazu ist denkbar einfach und lautet für unseren Fall `fenster.fill(SCHWARZ)`. Wenn Ihr eine andere Farbe als Hintergrund wollt – nur zu, Ihr wisst ja jetzt wie es geht.

Die zweite Verbesserung betrifft die Abfrage, ob die Schlange noch im Fenster ist, oder sich bereits außerhalb befindet. Dazu führen wir eine weitere `if`-Bedingung ein.

Wir kennen schon alle Parameter dafür, die Breite und Höhe des Fensters und die x- und die y-Position der Schlange (also des oberen linken Punktes des Quadrats, ich wollte es nur nochmal gesagt haben…).

Die Abfrage muss also folgendermaßen sein:

- Wenn die x-Position **kleiner als 0** ist, ist die Schlange links aus dem Fenster gekrochen.
- Wenn die x-Position **größer oder gleich der aktuellen Breite** des Fensters ist, ist die Schlange rechts aus dem Fenster gekrochen.
- Wenn die y-Position **kleiner als 0** ist, ist die Schlange oben aus dem Fenster gekrochen
- Wenn die y-Position **größer oder gleich der aktuellen Höhe** des Fensters ist, ist die Schlange unten aus dem Fenster gekrochen.

Und sollte eine der Bedingungen zutreffen, ist das Spiel verloren.

Also, der verbesserte Code in der Funktion `spielschleife_aufrufen()` sieht bei mir so aus:

```
72 …
73                 elif event.key == pygame.K_DOWN:
74                     snake_bewegung_x = 0
75                     snake_bewegung_y = quadrat_seite
76 
77         # Abfrage, ob die Schlange aus dem Fenster verschwindet.
78         # Wenn ja, dann hoert das Spiel auf
79         if snake_x >= BREITE or snake_x < 0 or snake_y >= HOEHE or snake_y < 0:
80             print('Verloren!')
81             spielen = False
82 
83         # Wenn nein, wird
84         # - die Veraenderung als neue Position gespeichert
85         # - und das Fenster wird neu aufgebaut
86         else:
87             snake_x += snake_bewegung_x
88             snake_y += snake_bewegung_y
89             fenster.fill(SCHWARZ)
90             pygame.draw.rect(fenster, ROT, [snake_x, snake_y, quadrat_seite, quadrat_seite])
91             pygame.display.update()
92             zeit.tick(snake_geschwindigkeit)
93 …
```

Zeit für ein kleines Fazit. Was haben wir bis hierhin programmiert?

- Wir haben ein Spielfenster definiert
- In diesem Fenster können wir ein Quadrat erscheinen lassen
- Mit Hilfe der Pfeiltasten, können wir das Quadrat in 4 Richtungen bewegen
- Kommt das Quadrat bei seiner Bewegung über die Abmessung des Fensters hinaus, wird das Spiel beendet
- Wenn der Spieler das X im Fenster drückt, wird das Spiel ebenfalls beendet

Schöner Erfolg, oder?

Probiert jetzt gerne Größen, Farben, Geschwindigkeit und Hintergründe aus, macht Euch mit dem Code bis hierhin vertraut.

4.3.6. Das Futter platzieren

Kommen wir zur Nahrung der Schlange.

Das Futter besteht aus Quadraten, die genauso groß sind, wie der Kopf der Schlange. In unserem Fall also 10 Pixel breit und 10 Pixel hoch. Um das Futter farblich von der Schlange zu unterscheiden, wählen wir grün.

Das Erscheinen des Futters im Fenster erfolgt genau wie das Erzeugen des Schlangenkopfes. Wir duplizieren also die Zeile

```
qc pygame.draw.rect(fenster, ROT, [snake_x, snake_y, quadrat_seite, quadrat_seite])
qec
```

und setzen statt „snake“ einfach „essen“ ein:

```
qc pygame.draw.rect(fenster, GRUEN, [essen_x, essen_y, quadrat_seite,
quadrat_seite]) qec
```

fertig.

Aber wie finden wir jetzt die Werte für `essen_x` und `essen_y`?

Da wird es jetzt etwas knifflig. Mit der Bibliothek `random()` haben wir ja ein Werkzeug, das uns Zufallszahlen aus einem vorher definierten Zahlenpool liefert, also eigentlich kein Problem.

Ich greife mal vor auf das nächste Kapitel. Wie prüfen wir, dass die Schlange das Essen gefunden hat?

Na, wenn `snake_x` und `essen_x` gleich sind UND wenn `snake_y` und `essen_y` gleich sind, dann ist die Schlange *genau über* dem Futter.

Und das ist auch schon der Knackpunkt. Die Schlange bewegt sich in 10er Schritten nach links, rechts, oben und unten. Sollte die Funktion `random()` die x-Position 142 ausgewählt haben, schaffen wir es nie, unseren Happen zu essen, weil wir entweder mit x = 140 oder x = 150 ankommen.

Blöd…

Ich habe mir nun überlegt, dass `random()` sich gleich aus 10er Schritten bedienen soll, sowohl für die x-Achse, als auch für die y-Achse. Für genau diese Art von Verarbeitung gibt es in Python die **Listen**.

Wir erstellen und also 2 Listen, eine für die Werte der x-Achse, und eine für die Werte der y-Achse.

Das lassen wir vor der Spielschleife tun, damit diese das ganze Spiel über zur Verfügung stehen. Dazu wird eine neue Funktion bereitgestellt (`futter_vorbereiten()`), den Code gibt es gleich unten.

Schauen wir uns also den Code dazu an:

```
1 ##################################################
2 #  Definition der Importe
3 ##################################################
4 import pygame
5 import random
6 …
```

Den Import haben wir ja schon besprochen.

Jetzt zur Deklaration der (globalen) Variablen:

```
36 …
37 # Groesse des Quadrats
38 quadrat_seite = 10
39
40 # Essen zubereiten
41 # Liste der x-Positionen
42 liste_essen_x = []
43 # Liste der y-Positionen
44 liste_essen_y = []
45 # Position von x
46 essen_x = 0
47 # Position von y
48 essen_y = 0
49 …
```

Zunächst haben wir die beiden Listen für die Positionen auf der x-Achse (Zeile 42) bzw. der y-Achse (Zeile 44).

Die konkreten Werte `essen_x` bzw. `essen_y` starten mit 0.

Nun zur Funktion `futter_vorbereiten()`. Sie sieht so aus:

```
53 …
54 def futter_vorbereiten():
55     print('in futter_vorbereiten')
56     e_x = 0
57     e_y = 0
58     for i in range (0, int(BREITE/quadrat_seite)):
59         liste_essen_x.append(e_x)
60         e_x += quadrat_seite
61     #print('liste_essen_x = ', liste_essen_x)
62     for j in range (0, int(HOEHE/quadrat_seite)):
63         liste_essen_y.append(e_y)
64         e_y += quadrat_seite
65     #print('liste_essen_y = ', liste_essen_y)
66 …
```

Was passiert darin?

Mit `e_x` und `e_y` haben wir 2 lokale(!) Variablen, die nur in dieser Funktion angesprochen werden.

Es folgt die Schleife zur Ermittlung der Werte für die x-Achse (ab Zeile 58). Mit `int(BREITE/quadrat_seite)` lassen wir das Programm selbst bestimmen, wann die Schleife zu Ende ist. Aktuell haben wir 800 / 10, also nach 80 Durchläufen ist Ende.

In der Schleife selbst addieren wir zum aktuellen Wert von `e_x` die Seitenlänge des Quadrats dazu (Zeile 59) und hängen den neuen Wert `e_x` an das Ende der Liste (Zeile 60).

Der erste Eintrag in der Liste ist also die 0, der zweite die 10 und so weiter, der 80. Eintrag ist dann 790.

Klar soweit, oder? Falls nicht, aktiviert mal die Ausgaben in der Zeile 61, dann wird das klar.

Die 2. Schleife erstellt analog die Liste für die Werte des y-Achse.

In der Spielschleife selbst, müssen wir auch Änderungen vornehmen:

```
75 …
76 def spielschleife_aufrufen():
77     print('in spielschleife_aufrufen')
78     global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y
79 
80     essen_x = random.choice(liste_essen_x)
81     essen_y = random.choice(liste_essen_y)
82 
83     while spielen:
84 …
```

Bevor es in die Schleife geht, besorgen wir uns eine x- und eine y-Koordinate aus den beiden erzeugten Listen.

Das Ausgeben des Futter-Quadrats machen wir direkt vor dem Ausgeben der Schlange:

```
114 …
115         else:
116             snake_x += snake_bewegung_x
117             snake_y += snake_bewegung_y
118             fenster.fill(SCHWARZ)
119             pygame.draw.rect(fenster, GRUEN, [essen_x, essen_y, quadrat_seite, quadrat_seite])
120             pygame.draw.rect(fenster, ROT, [snake_x, snake_y, quadrat_seite, quadrat_seite])
121             pygame.display.update()
122             zeit.tick(snake_geschwindigkeit)
123 …
```

Jetzt bleibt uns nur noch, die Funktion `futter_vorbereiten()` aufzurufen:

```
131 ##################################################
132 #  Programmstart
133 ##################################################
134 
135 print('Programmstart!')
136 
137 futter_vorbereiten()
138 
139 fenster = fenster_definieren()
140 
141 spielschleife_aufrufen()
142 
143 spiel_verlassen()
144 
```

Prima, damit sind wir auch mit der Erstellung des Futters und der Platzierung im Fenster durch.

Allerdings macht das jetzt noch nicht so viel Spaß, weil die Schlange das Futter nicht aufnimmt.

Genau das gehen wir im nächsten Kapitel an.

4.3.7. Die Schlange wächst

Jetzt brauche ich Eure Vorstellungskraft.

Das Wachsen der Schlange ist jetzt eine Mischung von allem, was wir in den vergangenen Kapiteln gemacht haben.

Wir merken uns, wie viele Quadrate wir bisher gegessen haben, und nur für diese Anzahl lassen wir eine Spur ausgeben, wie wir das am Ende von Kapitel „4.3.4 Die Bewegung der Schlange“ gemacht haben.

- Hat die Schlange also nur den Kopf, ist das, was wir anzeigen auch nur der Kopf.
- Hat die Schlange das erste Futter-Quadrat gegessen, zeigen wir den Kopf *und* das erste Quadrat an, wir sind also in Summe bei 2 Quadraten. Mit anderen Worten, die Länge der Spur ist dann 2.
- Bei 5 gegessenen Quadraten sind wir dann also bei einer Spur von 6 Quadraten, okay?

Wie schon im vorigen Kapitel, arbeiten wir jetzt auch hier mit einer Liste, in der wir die Spur speichern.

Gut, kommen wir dann zum Programmteil. Zuerst wieder die Deklaration der neuen globalen Variablen:

```
46 …
47 # Position von y
48 essen_y = 0
49
50 # Liste zur Speicherung des Kopfes und der gefressenen Quadrate
51 snake_als_liste = []
52 # Startwert der Länge der Schlange bei Spielbeginn ist 1 (der Kopf halt…)
53 snake_laenge = 1
54
55 ################################################
56 #  Definition der Funktionen
57 ################################################
58 …
```

Die Anpassungen erfolgen in der `spielschleife_aufrufen()`-Funktion. Dort zuerst wieder die Nutzung der globalen Variablen:

```
82 …
83     global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y, \
84            snake_laenge
85 …
```

Beachtet das Komma und den Backslash (‚\‘) am Ende der Zeile 83. Der Backslash ist das Zeichen für den Zeilenumbruch.

Die Veränderungen im `else`-Zweig der `if`-Abfrage zur Bestimmung, ob die Schlange aus dem Fenster gelaufen ist:

```
121         # Fenster wird neu aufgebaut
122         else:
123             snake_x += snake_bewegung_x
124             snake_y += snake_bewegung_y
125             fenster.fill(SCHWARZ)
126             pygame.draw.rect(fenster, GRUEN, [essen_x, essen_y, quadrat_seite, quadrat_seite])
127
128             # aktuelle Position des Kopfes in einer Liste speichern...
129             snake_kopf = []
130             snake_kopf.append(snake_x)
131             snake_kopf.append(snake_y)
132             #print('snake_als_liste = ', snake_als_liste)
133
134             # ... und an die bestehende Liste anhaengen
135             snake_als_liste.append(snake_kopf)
136
137             # der letzte Eintrag wird geloescht, damit der Eindruck entsteht,
138             # dass sich die Schlange bewegt
139             if len(snake_als_liste) > snake_laenge:
140                 del snake_als_liste[0]
141
142             # die Spur augeben
143             for snake in snake_als_liste:
144                 pygame.draw.rect(fenster, ROT, [snake[0], snake[1], quadrat_seite, \
```

```
145                                                 quadrat_seite])
146
147             # das Essen wurde gefangen, also erhoeht sich die Laenge der Schlange um ein
148             # Quadrat. Dann muss auch noch ein neues Futter-Quadrat platziert werden
149             if snake_x == essen_x and snake_y == essen_y:
150                 snake_laenge += 1
151                 essen_x = random.choice(liste_essen_x)
152                 essen_y = random.choice(liste_essen_y)
153
154             pygame.display.update()
155             zeit.tick(snake_geschwindigkeit)
156
```

Durch die Kommentare im Code sollte eigentlich klar sein, was passiert, oder?

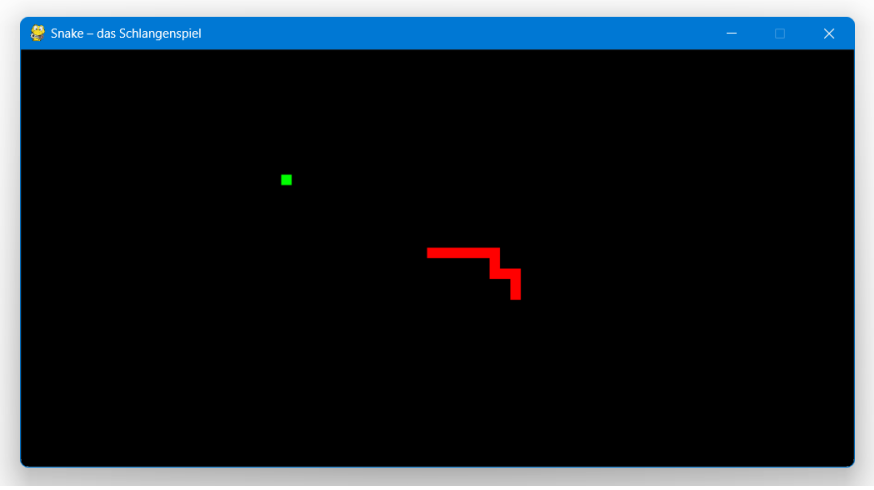
Die aktuelle Position des Kopfes der Schlange wird also bei jedem Durchlauf der Schleife in der lokalen Liste `snake_kopf` gespeichert, diese lokale Liste wird dann an die nächste, globale Liste `snake_als_liste` angehängt (Zeilen 129 bis 135).

Die Position des vormals letzten Quadrates wird gelöscht, sodass beim `update` des Fensters (Zeile 154) der Eindruck entsteht, dass die Schlange sich bewegt.

In Zeile 149 wird dann (wie im vorigen Kapitel angedeutet) geprüft, ob die beiden x- und y-Werte von Schlange und Essen übereinstimmen, dann ist das Fressen gefunden.

Die Länge der Schlange wird um 1 erhöht, sodass die Spur ab diesem Zeitpunkt (also für den nächsten Durchlauf) ein Quadrat mehr mit ausgibt.

Zum Ende des Kapitels lassen wir das ganze mal laufen. Bei mir sieht das dann so aus:



Läuft!

Prima, ich bin stolz auf Euch!

Momentan wissen wir aber noch nicht, wie lang die Schlange tatsächlich ist. Und wir müssen das Spiel auch immer wieder neu starten, das geht doch bestimmt besser, oder?

Klar…

4.3.8. Den Spielstand ausgeben

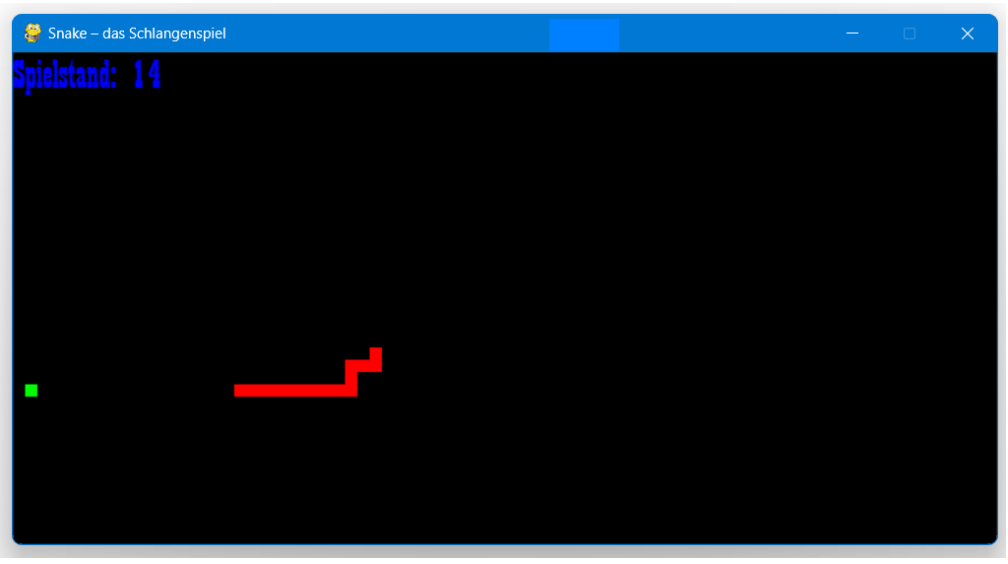
Bisher ist da Programm noch recht stumm.

Mit 2 einfachen Erweiterungen können wir einen Spielstand ausgeben:

```
153 …
154             # Ausgabe Spielstand - immer 1 weniger, als die Schlange lang ist
155             spielstand_anzeigen(snake_laenge - 1)
156
157             # aktualisieren des Fensters
158             pygame.display.update()
159             zeit.tick(snake_geschwindigkeit)
160
161
162 def spielstand_anzeigen(punktestand):
163     schrift_punktestand = pygame.font.SysFont("playbill", 35)
164     anzeige = schrift_punktestand.render("Spielstand: " + str(punktestand), True, BLAU)
165     fenster.blit(anzeige, [0, 0])
166
167
168 def spiel_verlassen():
169     print('in spiel_verlassen')
170     pygame.quit()
171     quit()
172 …
```

Bevor wir das Fenster aktualisieren, rufen wir eine neue Funktion `spielstand_anzeigen()` auf (Zeile 155), der wir den Punktestand übergeben.

In der Funktion selbst wird die Schriftart und -größe gesetzt (Zeile 163), der Schriftzug zu einem Bild zusammengesetzt und das Bild wird im Fenster platziert. Also eigentlich *liegt* das Bild *auf dem Fenster*, aber das ist nicht so wichtig. Wichtiger ist, dass wir ab sofort eine Punktestand haben – yeah!



Es gibt über 300 verschiedenen Schriftarten in `pygame`. Da es mir unmöglich erschien, alle einzeln aufzurufen, habe ich mir zur Auswahl ein kleines Hilfsprogramm geschrieben.

Mit Hilfe der Pfeiltasten kann ich mich durch die einzelnen Schriftarten navigieren und sehe sofort, wie das im Spiel aussieht. Den Code dazu habe ich auch auf der Homepage abgelegt, sucht mal unter dem Stichwort „Helferlein“.

Bleibt noch eine Sache zu tun, wenn das Spiel verloren ist, soll der Spieler sich entscheiden können, ob er noch einmal spielen möchte, oder das Spiel beenden will.

Aktuell machen wir das Spiel immer sofort zu, das müssen wir ändern und darum kümmern wir uns im letzten Kapitel.

4.3.9. Die Wiederholung

Die Ausgabe einer Nachricht geht im Wesentlichen analog der Anzeige des Spielstands. Im Gegensatz zu diesem, wird die Nachricht aber nur am Ende des Spiels ausgegeben und nicht permanent aktualisiert.

Zuerst brauchen wir die neue globale boolsche Variable `verloren`. Initial setzen wir sie auf `False`.

```
14 …
15 # Spielsteuerung
16 spielen = True
17 verloren = False
18 …
```

Als nächstes ändern wir die Abfrage, ob die Schlange aus dem Fenster gekrochen ist. Die Ausgabe „Verloren!“ löschen wir und statt `spielen = False` setzen wir die neue globale Variable `verloren` auf `True`:

```
140 …
141         # Abfrage, ob die Schlange aus dem Fenster verschwindet.
142         # Wenn ja, wird die Variable ‚verloren‘ auf True gesetzt
143         if snake_x >= BREITE or snake_x < 0 or snake_y >= HOEHE or snake_y < 0:
144             verloren = True
145 …
```

Jetzt springe ich etwas - unter der im letzten Kapitel angelegten Funktion `spielstand_anzeigen()`, nehmen wir eine weitere Funktion auf und nennen sie `nachricht_anzeigen()`:

```
187 …
188 def spielstand_anzeigen(punktestand):
189     schrift_punktestand = pygame.font.SysFont("playbill", 35)
 10     anzeige  = schrift_punktestand.render("Spielstand: " + str(punktestand), True, BLAU)
191     fenster.blit(anzeige, [0, 0])
192
193
194 def nachricht_anzeigen(nachricht, farbe):
195     font_style = pygame.font.SysFont("arial", 25)
196     bild = font_style.render(nachricht, True, farbe)
197     fenster.blit(bild, [180, HOEHE/2])
198 …
```

Als Übergabeparameter erwartet die Funktion 2 Strings, die Nachricht selbst und eine Farbe für die Schrift. In der Funktion wird zuerst die Schriftart und die -größe an eine lokale Variable übergeben, ich habe mich für „Arial“ in Größe „25“ entschieden. Nehmt gerne etwas anderes!

Die Nachricht und der Text werden zu einem Bild „*gerendert*“, das dann im Fenster angezeigt wird, an Position 180 Pixel auf der x-Achse und der Wert der Hälfte des Bildschirms auf der y-Achse.

Direkt unterhalb der Schleife `while spielen` bauen wir noch eine neue Schleife ein, `while verloren` (Achtung, das ist *in* der `while spielen`-Schleife, also eine *Schleife in der Schleife*!):

```
 81 …
 82 def spielschleife_aufrufen():
 83     print('in spielschleife_aufrufen')
 84     global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y, \
 85            snake_laenge, verloren, snake_als_liste
 86 
 87     essen_x = random.choice(liste_essen_x)
 88     essen_y = random.choice(liste_essen_y)
 89 
 90     while spielen:
 91 
 92         # dieser Teil wird nur aufgerufen, wenn das Spiel verloren wurde
 93         while verloren:
 94             fenster.fill(BLAU)
 95             nachricht_anzeigen("Verloren! Nochmal mit \"c\", verlassen mit \"q\" ", ROT)
 96             pygame.display.update()
 97 
 98             for event in pygame.event.get():
 99                 if event.type == pygame.KEYDOWN:
100                     if event.key == pygame.K_q:
101                         spielen = False
102                         verloren = False
103                     if event.key == pygame.K_c:
104                         spielen = True
105                         verloren = False
106                         snake_x = BREITE/2
107                         snake_y = HOEHE/2
108                         snake_bewegung_x = 0
109                         snake_bewegung_y = 0
110                         snake_als_liste = []
111                         snake_laenge = 1
112                         futter_vorbereiten()
113                         spielschleife_aufrufen()
114 
115                 if event.type == pygame.QUIT:
116                     spielen = False
117                     verloren = False
118 
119             for event in pygame.event.get():
120 …
```

Sobald `verloren` auf `True` gesetzt wurde, springt die Verarbeitung hier hin.

Das Fenster wird blau (Zeile 94), die Nachricht wird durch Aufruf der Funktion ausgegeben und es wird auf eine Entscheidung des Spielers in Form einer Eingabe gewartet.

Entscheidet sich der Spieler für das Drücken der q-Taste (Zeile 100), wird `spielen` auf `False` gesetzt, damit die `while spielen`-Schleife ihre Abbruchbedingung bekommt und `verloren` wird auf `False` gesetzt, damit wir die `while verloren`-Schleife verlassen können.

Gleiches geschieht, wenn der Spieler das X am oberen rechten Fensterrand klickt (Zeile 115).

Entscheidet sich der Spieler hingegen für das Drücken der c-Taste (Zeile 103) bleibt `spielen` explizit auf `True`, `verloren` wird zurückgesetzt auf `False` und alle für den Neustart notwendigen Grundwerte werden zurückgesetzt. Am Ende wird ein neues Futter-Quadrat erzeugt (Zeile 112) und die Spielschleife wird erneut aufgerufen.

Damit geht das Spiel von vorne los.

4.3.10. Der finale Stand

Zum Abschluss hier noch einmal der finale Code als Version mit auskommentierten oder gelöschten `print()`-Anweisungen:

```
 1 ##################################################
 2 #  Definition der Importe
 3 ##################################################
 4 import pygame
 5 import random
 6
 7 ##################################################
 8 #  Definition der Konstanten und Variablen
 9 ##################################################
10
11 #  Werte für das Spielefenster
12 BREITE = 800
13 HOEHE = 400
14
15 # Spielsteuerung
16 spielen = True
17 verloren = False
18
19 # Farben RGB (Red, Green, Blue), jeweils 0 - 255
20 ROT = (255,0,0)
21 GRUEN = (0,255,0)
22 BLAU = (0,0,255)
23 WEISS = (255, 255, 255)
24 SCHWARZ = (0, 0, 0)
25
26 # Positionen der oberen linken Ecke des Kopfes
27 snake_x = BREITE/2
28 snake_y = HOEHE/2
29
30 # Bewegungen auf der x- und der y-Achse
31 snake_bewegung_x = 0
32 snake_bewegung_y = 0
33
34 # Geschwindigkeiten
35 zeit = pygame.time.Clock()
36 snake_geschwindigkeit = 20
37
```

```
38 # Groesse des Quadrats
39 quadrat_seite = 10
40
41 # Essen zubereiten
42 # Liste der x-Positionen
43 liste_essen_x = []
44 # Liste der y-Positionen
45 liste_essen_y = []
46 # Position von x
47 essen_x = 0
48 # Position von y
49 essen_y = 0
50
51 # Liste zur Speicherung des Kopfes und der gefressenen Quadrate
52 snake_als_liste = []
53 # Startwert der Länge der Schlange bei Spielbeginn ist 1 (der Kopf halt…)
54 snake_laenge = 1
55
56 ##################################################
57 #  Definition der Funktionen
58 ##################################################
59
60 def futter_vorbereiten():
61     # print('in futter_vorbereiten')
62     e_x = 0
63     e_y = 0
64     for i in range (0, int(BREITE/quadrat_seite)):
65         liste_essen_x.append(e_x)
66         e_x += quadrat_seite
67
68     for i in range (0, int(HOEHE/quadrat_seite)):
69         liste_essen_y.append(e_y)
70         e_y += quadrat_seite
71
72
73 def fenster_definieren():
74     # print('in fenster_definieren')
75     pygame.init()
76     f = pygame.display.set_mode((BREITE,HOEHE))
77     pygame.display.set_caption('Snake – das Schlangenspiel')
78     return f
79
80
81 def spielschleife_aufrufen():
82     # print('in spielschleife_aufrufen')
83     global spielen, snake_x, snake_y, quadrat_seite, snake_bewegung_x, snake_bewegung_y, \
84            snake_laenge, verloren, snake_als_liste
85
86     essen_x = random.choice(liste_essen_x)
87     essen_y = random.choice(liste_essen_y)
88
89     while spielen:
90
91         # dieser Teil wird nur aufgerufen, wenn das Spiel verloren wurde
92         while verloren:
93             fenster.fill(BLAU)
94             nachricht_anzeigen("Verloren! Nochmal mit \"c\", verlassen mit \"q\" ", ROT)
```

```
95              pygame.display.update()
96
97              for event in pygame.event.get():
98                  if event.type == pygame.KEYDOWN:
99                      if event.key == pygame.K_q:
100                         spielen = False
101                         verloren = False
102                     if event.key == pygame.K_c:
103                         spielen = True
104                         snake_x = BREITE/2
105                         snake_y = HOEHE/2
106                         snake_bewegung_x = 0
107                         snake_bewegung_y = 0
108                         snake_als_liste = []
109                         snake_laenge = 1
110                         verloren = False
111                         futter_vorbereiten()
112                         spielschleife_aufrufen()
113
114                 if event.type == pygame.QUIT:
115                     spielen = False
116                     verloren = False
117
118
119         for event in pygame.event.get():
120             # Wenn das x im Fenster gedrückt wird, soll das Fenster zugehen
121             if event.type == pygame.QUIT:
122                 spielen = False
123
124             # die Tastatur abfragen
125             if event.type == pygame.KEYDOWN:
126                 if event.key == pygame.K_RIGHT:
127                     snake_bewegung_x = quadrat_seite
128                     snake_bewegung_y = 0
129                 elif event.key == pygame.K_LEFT:
130                     snake_bewegung_x = quadrat_seite * -1
131                     snake_bewegung_y = 0
132                 elif event.key == pygame.K_UP:
133                     snake_bewegung_x = 0
134                     snake_bewegung_y = quadrat_seite * -1
135                 elif event.key == pygame.K_DOWN:
136                     snake_bewegung_x = 0
137                     snake_bewegung_y = quadrat_seite
138
139         # Abfrage, ob die Schlange aus dem Fenster verschwindet.
140         # Wenn ja, wird die Variable 'verloren' auf True gesetzt
141         if snake_x >= BREITE or snake_x < 0 or snake_y >= HOEHE or snake_y < 0:
142             verloren = True
143
144         # Wenn nein, wird
145         # - die Veraenderung als neue Position gespeichert
146         # Fenster wird neu aufgebaut
147         else:
148             snake_x += snake_bewegung_x
149             snake_y += snake_bewegung_y
150             fenster.fill(SCHWARZ)
151             pygame.draw.rect(fenster, GRUEN, [essen_x, essen_y, quadrat_seite, quadrat_seite])
```

```
152
153             # aktuelle Position des Kopfes in einer Liste speichern...
154             snake_kopf = []
155             snake_kopf.append(snake_x)
156             snake_kopf.append(snake_y)
157
158             # ... und an die bestehende Liste anhängen
159             snake_als_liste.append(snake_kopf)
160
161             # der letzte Eintrag wird gelöscht, damit der Eindruck entsteht,
162             # dass sich die Schlange bewegt
163             if len(snake_als_liste) > snake_laenge:
164                 del snake_als_liste[0]
165
166             # die Spur augeben
167             for snake in snake_als_liste:
168                 pygame.draw.rect(fenster, ROT, [snake[0], snake[1], quadrat_seite, \
169                                                 quadrat_seite])
170
171             # das Essen wurde gefangen, also erhöht sich die Länge der Schlange um ein
172             # Quadrrat. Dann muss auch noch ein neues Futter-Quadrat platziert werden
173             if snake_x == essen_x and snake_y == essen_y:
174                 snake_laenge += 1
175                 essen_x = random.choice(liste_essen_x)
176                 essen_y = random.choice(liste_essen_y)
177
178             # Ausgabe Spielstand ist immer einer weniger, als die Schlange lang ist
179             spielstand_anzeigen(snake_laenge - 1)
180
181             # aktualisieren des Fensters
182             pygame.display.update()
183             zeit.tick(snake_geschwindigkeit)
184
185
186 def spielstand_anzeigen(punktestand):
187     # print('in spielstand_anzeigen')
188     schrift_punktestand = pygame.font.SysFont("playbill", 35)
189     anzeige  = schrift_punktestand.render("Spielstand: " + str(punktestand), True, BLAU)
190     fenster.blit(anzeige, [0, 0])
191
192
193 def nachricht_anzeigen(nachricht, color):
194     # print('in nachricht_anzeigen')
195     font_style = pygame.font.SysFont("arial", 25)
196     text = font_style.render(nachricht, True, color)
197     fenster.blit(text, [180, HOEHE/2])
198
199
200 def spiel_verlassen():
201     # print('in spiel_verlassen')
202     pygame.quit()
203     quit()
204
205
206 ###################################################
207 #  Programmstart
208 ###################################################
```

```
209
210 # print('Programmstart!')
211
212 futter_vorbereiten()
213
214 fenster = fenster_definieren()
215
216 spielschleife_aufrufen()
217
218 spiel_verlassen()
219
```

5. Abschluss

So, damit bin ich auch schon wieder am Ende des Projekts angelangt. Ich hoffe, es hat Euch ebenso viel Spaß gemacht wie mir.

Was gibt es noch zu tun? Raum für Verbesserungen gibt es immer. Aber 2 Fehler haben wir noch im Code, die will ich Euch nicht verschweigen.

Die Schlange kann vorwärts und rückwärts laufen. Das ist im Original nicht möglich, soweit ich das weiß. Technisch kann man das abfangen, wenn man permanent die Position des ersten Futter-Quadrats mit der des Kopfes vergleicht. Wenn die beiden x- und y-Koordinaten gleich sind, geht die Schlange rückwärts. Das wäre der Punkt, an dem wir `spielen` wieder auf `False` setzen könnten.

Den 2. Fehler werdet Ihr finden, wenn Ihr die Variable `quadrat_seite` mal von 10 auf zum Beispiel 15 ändert. Ihr werdet vermutlich keinen Punkt machen können, da wir immer nur in 10er Schritten vorwärts gehen und (die obere linke Ecke von) `snake` und (die obere linke Ecke von) `essen` nur wirklich zufällig gleich sind. Probiert das aber unbedingt mal selber aus!

Auch dieses Problem kann man technisch lösen, wir müssten mit Näherungswerten arbeiten. Also `snake_x` und `essen_x` müssten weniger als `quadrat_seite` auseinander sein, dann haben sich die Flächen zumindest berührt. Vorausgesetzt `snake_y` und `essen_y` sind ähnlich weit voneinander entfernt.

Was kann man aber sonst noch verbessern? Hier einige unsortierte Ideen, die ich aber nicht weiterverfolgen werde:

- Man könnte dem Spieler ermöglichen, seinen Namen einzugeben
- Den absoluten Highscore könnte man in eine Datei schreiben und bei Programmstart oben rechts anzeigen. Wenn es einen neuen Highscore gibt, müsste der dann in die Datei geschrieben werden
- Die Geschwindigkeit der Schlange könnte mit zunehmender Punktezahl steigen
- Die Schlange, das Futter und der Hintergrund könnte man durch Bilder in Form von png-Dateien ersetzen

Das überlasse ich aber Eurer Fantasie. Ansonsten gilt wie immer, viel Spaß beim Nachbauen und wenn Euch was Tolles einfallen sollte, was man umsetzen könnte, schickt mir gerne eine Mail an papa@papa-programmiert.de, ich würde mich sehr freuen, genau wie auch über eine Rückmeldung zu diesem Dokument.

Viel Spaß weiterhin beim Coden, bleibt neugierig und hartnäckig.

Viele Grüße, Papa