

HI-SCORE
4525150

THOMAS KAFFKA

2UP
03647



RETRO- GAMES MIT PYTHON

**SPIELE-KLASSIKER
PROGRAMMIEREN
INKL. KI-FUNKTIONEN**



mitp

Inhaltsverzeichnis

	Einleitung	9
1	Schlangenbiss	13
1.1	Die Spielidee	13
1.2	Vorbereitungen für das Programm	14
1.2.1	Der Pixelaufbau	14
1.2.2	Farben	15
1.2.3	pygame	16
1.2.4	Der Game-Loop	17
1.3	Das Programm	17
1.4	Das Spiel spielen	29
2	Press Under Water	31
2.1	Die Spielidee	31
2.2	Das Programm	32
2.3	Das Spiel spielen	50
3	Pyramide	51
3.1	Die Spielidee	51
3.2	Vorbereitung für das Programm	52
3.3	Das Programm	53
3.3.1	Das Hauptprogramm	53
3.3.2	Die Klasse Globals	69
3.3.3	Die Klasse Brick	75
3.4	Das Spiel spielen	77
4	Tetrimania	79
4.1	Die Spielidee	79
4.2	Vorbereitungen für das Programm	80
4.3	Das Programm	81
4.3.1	Das Hauptprogramm	81
4.3.2	Die Elternklasse Piece	97
4.3.3	Die Klasse L_Piece	101
4.4	Das Spiel spielen	102

5	Diamond Pusher	103
5.1	Die Spielidee	103
5.2	Vorbereitungen für das Programm	104
5.3	Das Programm	105
5.3.1	Das Hauptprogramm	105
5.3.2	Die Klasse Globals	118
5.3.3	Die Klasse Levelmanager	120
5.3.4	Die Klasse Avatar	125
5.3.5	Die Klasse Push	132
5.4	Das Spiel spielen	135
6	Arcade Fighter	137
6.1	Die Spielidee	137
6.2	KI für das Spiel	137
6.3	Neuronale Netze	138
6.4	Das Programm klassisch	141
6.5	KI-Ergänzungen	158
6.5.1	Training der Aliens	158
6.5.2	KI-Ergänzungen im Spielprogramm	161
6.6	Das Spiel spielen	166
7	Robot Chase	167
7.1	Die Spielidee	168
7.2	Vorbereitungen für das Programm	168
7.2.1	Die Arena	169
7.2.2	Das Q-Learning-Konzept	170
7.2.3	Q-Learning formalisiert	171
7.3	Das Programm	172
7.3.1	Das Hauptprogramm	172
7.3.2	Die Elternklasse Character	194
7.3.3	Die Klasse Avatar	196
7.3.4	Die Klasse Robot	198
7.3.5	Die Klasse QLearning	201
7.4	Das Spiel spielen	210
8	Landgewinn	211
8.1	Die Spielidee	211
8.2	Vorbereitungen für das Programm	212

8.3	Das Programm	213
8.3.1	Das Hauptprogramm	213
8.3.2	Die Klasse Character	230
8.3.3	Die Klasse Player	237
8.3.4	Die Klasse Computer	238
8.4	Das Spiel spielen	244
A	Mit Python programmieren	245
A.1	Diktion	246
A.2	Erste Schritte	247
A.3	Programmschleifen	249
A.3.1	Fibonacci-Zahlen	253
A.4	Programmbedingungen	255
A.5	Zeichenketten (Strings)	256
A.5.1	Palindrom-Test	260
A.6	Zahlen	261
A.6.1	Fakultät	264
A.7	Listen	266
A.7.1	Die Liste	266
A.7.2	Sieb des Eratosthenes	268
A.7.3	Das Tupel	270
A.7.4	Das Set	270
A.7.5	Das Dictionary	271
A.8	Funktionen	272
A.9	Nimm-Spiel	274
A.10	Objektorientierte Programmierung	278
	Stichwortverzeichnis	281

Einleitung

Du möchtest Python erlernen? Vielleicht ist das ja dein Einstieg in die Programmierung und du kennst noch keine andere Programmiersprache. Dann ist mein Buch genau das Richtige für dich! Du wirst Python anhand von verschiedenen Computerspielen lernen. Bei den Spielen handelt es sich um Retrospiele, also Spiele, wie sie früher (aber auch heute noch) auf Computern gespielt wurden. Die Spiele sind durchweg zweidimensional, aber trotzdem sehr spannend. Dadurch sind sie aber nicht so aufwändig zu programmieren und du erhältst einen guten Einstieg in das Programmieren.

Bei der Auswahl und der Darstellung der Spiele bin ich so vorgegangen, dass die leichten Spiele zuerst vorgestellt werden. Wenn du also das Buch Schritt für Schritt bearbeitest, werden die Programme immer anspruchsvoller.

Das Buch ist wie folgt aufgebaut:

In *Kapitel 1* stelle ich das Spiel **Schlangenbiss** vor. Es geht darum, eine Schlange so über den Bildschirm zu steuern, dass sie einen Apfel frisst, aber nicht in die Hindernisse oder in sich selbst kriecht. Wenn die Schlange einen Apfel isst, wird sie ein Segment länger, das Spiel wird also immer schwieriger.

Kapitel 2 behandelt das Spiel **Press Under Water**. Im Spiel werden Bilder angeklickt, um sie unter Wasser zu drücken und sie so aus dem Spiel zu entfernen. Alle angrenzenden gleichen Bilder werden mit entfernt. Das Spiel beinhaltet auch eine Punkte Verarbeitung.

Das *Kapitel 3* stellt das Spiel **Pyramide** vor. Es handelt sich um einen Mahjongg Klon. Es wird mit Spielsteinen eine (pseudo)dreidimensionale Pyramide gebildet, deren Randsteine mit Klicks entfernt werden können.

Im *Kapitel 4* gehe ich auf das Spiel **Tetrimania** ein. Es ist ein Tetris-Klon. Das Spiel besitzt eine Highscore Verarbeitung und es wird eine Hintergrundmusik abgespielt.

Kapitel 5 bezieht sich auf das Spiel **Diamond Pusher**. Es ist ein sehr anspruchsvolles Spiel, was eine Menge Spaß macht. Es geht darum, in einem Labyrinth Diamanten auf deren Fassungen zu verschieben. Was sich sehr einfach anhört, ist in der Praxis dann aber sehr komplex bis fast unlösbar.

In *Kapitel 6* behandle ich das Spiel **Arcade Fighter**. Es ist das erste Spiel, in welchem KI eingesetzt wird. Du steuerst eine Rakete, die auf sie zufliegende Asteroiden und Alien-Raumschiffe abschießen soll. Die Aliens schießen immer nur, wenn dein Raumschiff in Reichweite ist. Dazu wird ein »neuronales Netz« eingesetzt.

Kapitel 7 macht dich mit dem Konzept Q-Learning vertraut. Es handelt sich um das Spiel **Robo Chase**. Roboter jagen dich durch ein Labyrinth aus Blitzen. Du kannst dich so bewegen, dass du die Roboter in die Blitze lenkst. Im Anschluss daran führen wir ein Experiment durch, bei welchem ein Roboter den Weg zu deinem Avatar findet, indem Q-Learning eingesetzt wird.

Und im *Kapitel 8*, welches das Spiel **Landgewinn** beschreibt, wird der Mini-Max Algorithmus zur Ermittlung des für den Computer besten Zugs eingesetzt. Das Spiel ist ein Reversi-Klon und kann zufällig, mit einfacher Punkteermittlung und vorausschauend gespielt werden. Die dritte Spielvariante ist die stärkste.

Installationen

Zunächst benötigst du eine Version von Python auf deinem Computer. Diese holst du dir von der Webseite <https://www.python.org/downloads/> und installierst sie anhand der Vorgaben des Installationsprogramms.

Dann möchtest du später Programmtext erfassen und ausführen. Dazu benötigst du eine Integrierte Entwicklungsumgebung (IDE). Ich empfehle dir dazu Spyder. Hole dir die neuste Version von der Webseite <https://docs.spyder-ide.org/current/installation.html> und installiere sie. Du kannst natürlich auch eine andere IDE verwenden.

Damit du mit Python Spiele programmieren kannst, ist noch die Bibliothek `pygame` erforderlich. Diese installierst du am besten mit dem Installationsprogramm von Python: `pip`. Dazu rufst du die Eingabeaufforderung (die findest du unter **Windows / Tools**) auf und gibst den folgenden Befehl ein:

```
pip install pygame
```

Dann musst du evtl. bei Linux `tkinter` installieren:

```
sudo apt install python3-tk
```

Für die neuronalen Netze musst du folgende Bibliotheken installieren (du brauchst die Bibliotheken im Internet nicht zu suchen, sie werden mit `pip` automatisch installiert):

```
pip install pandas
```

```
pip install joblib
```

```
pip install tensorflow
```

```
pip install scikit-learn
```

```
pip install numpy (wurde bereits bei pandas installiert)
```

Damit hast du alle Vorbereitungen abgeschlossen.

Was du noch wissen musst

Bevor wir beginnen, möchte ich ein paar der wichtigsten Dinge ansprechen.

Die im Buch behandelten Programme kannst du dir von der Seite

www.mitp.de/1178

downloaden. Wenn du dir ein Programm erarbeitest, hole es dir in deine IDE. Du wirst dabei feststellen, dass die Zeilennummern des Buchs von den Zeilennummern der Listings abweichen. Ich habe im Buch, der Übersichtlichkeit halber, auf Leerzeilen in den Listings weitestgehend verzichtet. Wenn im Text Zeilennummern angegeben werden, dann beziehen sie sich immer auf die des Buchs.

Es ist grundsätzlich eine gute Idee, Programme selbst zu erfassen und sich nicht auf die downloadbaren Programme zu beziehen. Wenn du Programme selbst erfasst, lernst du viel mehr.

Im Python-Sprachgebrauch werden Programme als »Skripte« bezeichnet. Um aber mit anderen Programmiersprachen übereinzustimmen, verwende ich immer die Bezeichnung »Programm«.

Wenn du dich in Python einarbeiten möchtest, sieh dir das Kapitel im Anhang an, bevor du mit den Spielprogrammen beginnst.

Der Game-Loop

Sämtliche Spiele werden durch einen sogenannten Game-Loop gesteuert. Es handelt sich um eine Endlosschleife, die mit dem Statement

```
while True:
```

gebildet wird.

Jetzt wirst du dich fragen, warum eigentlich eine Endlosschleife? Die endet ja nie! Das ist grundsätzlich richtig. Die Schleife soll in jeder Sekunde die Verarbeitung des Spieles mehrmals wiederholen. Das wird mit der Konstante FPS (Frames per

Second) gesteuert. Ein Frame ist die Anzeige eines Bildes des Spiels. Damit das Spiel aber beendet werden kann, müssen die Statements

```
pygame.quit()
```

```
sys.exit()
```

kontrolliert verarbeitet werden. Das erste Statement beendet pygame, das zweite das Fenster und damit den Game-Loop. Wenn du einen Fehler im Programm hast, hängt sich Python manchmal auf. Du kannst das Programm dann beenden, indem du auf den roten Button () in der oberen Fensterleiste klickst.

Schlangenbiss

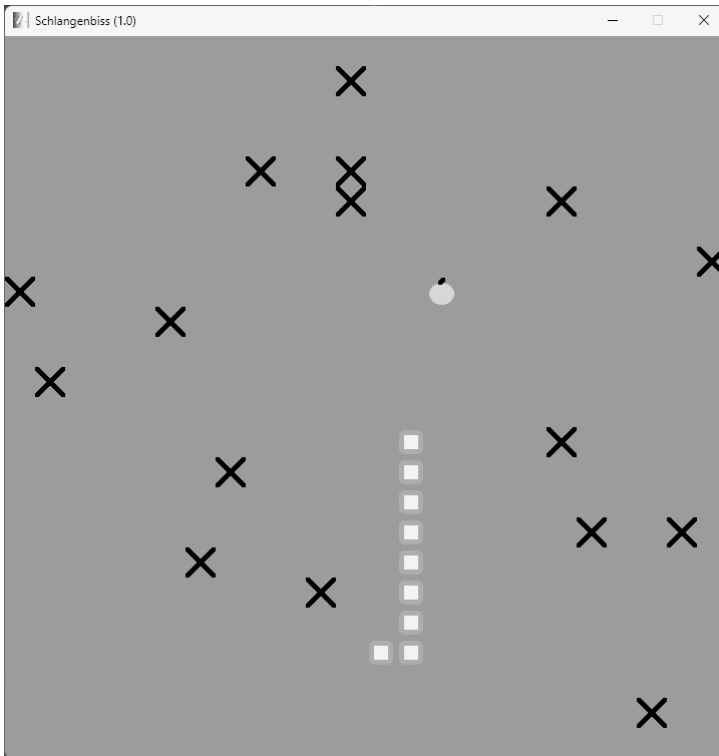


Abb. 1.1: Das Spiel Schlangenbiss

1.1 Die Spielidee

Das Spiel Schlangenbiss ist ein Clone des Retrospiels »Snake«. Dieses wurde 1976 noch mit dem damaligen Namen »Blockade« erfunden. Du steuerst eine Schlange, die einen zufällig erscheinenden Apfel essen soll. Bei jedem Bissen wird die Schlange ein Segment länger. Es ist wichtig, den Hindernissen auszuweichen und weder in den Rand noch in die eigene Schlange zu kriechen. Dies wird immer schwieriger, je länger die Schlange ist.

1.2 Vorbereitungen für das Programm

Zunächst möchte ich auf den Umstand eingehen, dass ich immer in englischer Sprache programmiere. Das hat einen bestimmten Grund.

Natürliche Sprache in Programmen

In Python-Programmen werden Kommentare (beginnend mit dem Zeichen #) verwendet, um den Programmcode zu erläutern. Es werden Variablen und Konstanten benannt sowie Funktionen. Jetzt fragt sich natürlich jeder, in welcher Sprache soll das geschehen. Ich bin dafür, sich anzugewöhnen, in Englisch zu programmieren, damit die eigenen Programme international verstanden werden. Dazu eine kurze Anekdote:

Als junger Programmierer sollte ich ein PPS-System weiterentwickeln und auf den deutschen Markt anpassen. Das System ist von einem dänischen Ableger des Softwarehauses erstellt worden, in dem ich damals arbeitete, und in dänischer Sprache ausgestaltet worden. Jetzt spreche ich leider kein Dänisch und daher war die ganze Sache für mich und jeden anderen in unserem Entwicklerbüro aussichtslos. Das hatte mich damals dazu gebracht, immer nur in Englisch zu programmieren.

1.2.1 Der Pixelaufbau

Für die Programme meines Buches ist es wichtig zu wissen, wie ein Fenster aufgebaut ist. Wenn du ein Vergrößerungsglas an deinen Computer-Bildschirm hältst, siehst du, dass das gesamte Bild durch Pixel (kleine farbige Punkte) aufgebaut wird. Wenn wir jetzt in einem Programmfenster Bilder verwenden, sind diese auch durch Pixel aufgebaut. Dabei ist es wichtig zu wissen, dass die einzelnen Pixel durch Koordinaten angesprochen werden können. Diese Koordinaten werden von links nach rechts beginnend mit der Null gezählt, sowie von oben nach unten, auch beginnend mit der Null. Also anders als im gewohnten Koordinatensystem, was die Y-Achse von unten nach oben orientiert. Das sieht dann vergrößert, wie in Abbildung 1.2 gezeigt wird, aus.

Wenn wir die Konvention einhalten, zunächst die Zeilenkoordinate zu nennen, dann finden wir schwarze Pixel an den Koordinaten (2, 4) sowie (7, 2). Wenn ich Variablen verwende, um die Koordinaten zu adressieren, verweise ich immer mit y auf die Zeile und mit x auf die Spalte bzw. mit i auf die Zeile und mit j auf die Spalte.

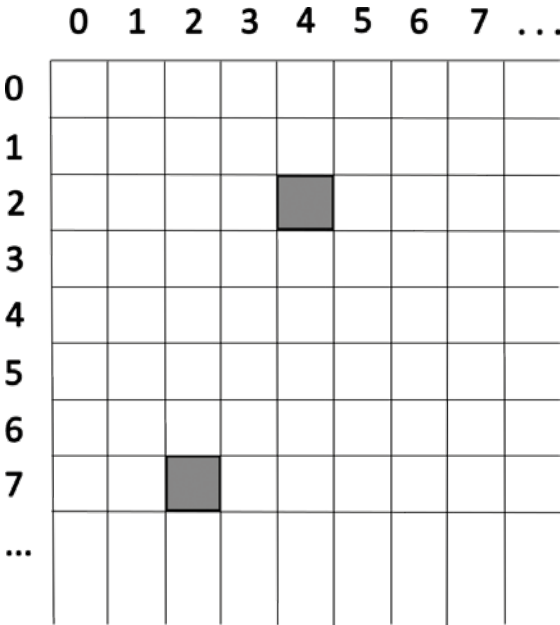


Abb. 1.2: Pixelaufbau der Abbildungen

1.2.2 Farben

Pixel können auch farbig dargestellt werden, indem jedem Pixel ein Tupel von drei Zahlen zugeordnet wird. Die erste Zahl bezieht sich auf den Rot-, die zweite auf den Grün- und die dritte auf den Blau-Anteil der Farbe. In einer solchen Weise codierte Farben werden auch »RGB-Farben« genannt.

Die Farb-Tupel unserer beiden Pixel von oben sind für Schwarz (0, 0, 0). Ein Zahlenwert kann maximal 255 sein, damit lassen sich dann $255 * 255 * 255 \sim 16$ Mio. Farben definieren. In der nächsten Tabelle sind einige Farbcodes aufgeführt.

Rot	Grün	Blau	Farbe
255	0	0	Rot
0	255	0	Grün
0	0	255	Blau
128	128	128	Grau
128	128	0	Olive
255	255	0	Gelb

Tabelle 1.1: Beispiele für Farbcodes

Über sehr viel mehr verschiedener RGB-Farben kannst du dich mit Hilfe der Webseite <https://www.farb-tabelle.de/de/farbtabelle.htm> informieren.

1.2.3 pygame

In meinem Buch entwickeln wir gemeinsam 2D-Spiele. Wir können solche Spiele auch als »Retro«-Spiele bezeichnen, da es sich dabei um die ersten, gängigen Computerspiele handelte. Um solche Spiele zu erstellen, gibt es für Python eine bestimmte, sehr nützliche Bibliothek mit dem Namen **pygame**. Mit **pygame** kann ein Spiel gesteuert werden und es können Sounds abgespielt und Bilder auf einem Programmfenster ausgegeben werden. **pygame** hat verschiedene Objekte, die dazu verwendet werden können.

Mit dem Statement

```
pygame.init()
```

wird **pygame** initialisiert.

Die Oberfläche des Fensters, auf der Grafik ausgegeben wird, wird definiert mit folgendem Statement.

```
SURFACE = pygame.display.set_mode((WINWIDTH, WINHEIGHT), 0, 32)
```

Dabei geben die Konstanten **WINWIDTH** und **WINHEIGHT** die Breite und Höhe des Fensters an. Die Null bedeutet, dass keine Flags gesetzt werden, und die 32, dass es sich um eine Farbbittiefe von 32 Bits handelt.

Hinweis zu Flags

Ein Flag ist in der Programmierung eine Variable, die nur eine Null oder eine 1 bzw. False oder True beinhalten kann. Es wird dazu benutzt, im Programm einen Hinweis zu geben, dass ein bestimmter Status oder eine bestimmte Situation vorliegt. Dann hat die Variable den Wert 1. Es wird gewissermaßen eine Flagge (engl. Flag) erhoben.

Ein Font (Schriftart), der zur Ausgabe von Text verwendet wird, wird so definiert.

```
FONT = pygame.font.SysFont('Arial', 24)
```

'Arial' ist dabei die Font-Familie und die 24 gibt an, dass der Font eine Größe von 24 Pixeln haben soll. Die zeitliche Steuerung des Spiels geschieht über das Statement

```
fpsClock = pygame.time.Clock()
```

Die damit definierte Uhr ermittelt, wie viele Millisekunden seit dem letzten Aufruf vergangen sind. Die Uhr wird im Game-Loop verwendet, um das Spiel auf der Basis einer bestimmten Framerate ablaufen zu lassen.

```
fpsClock.tick(FPS)
```

hält den Game-Loop so lange an, sodass das Spiel mit einer Framerate von FPS (beispielsweise 30) verarbeitet wird. Dabei bedeutet FPS »Frames per Second«. Mit dem Statement

```
SURFACE.fill(BGCOLOR)
```

wird das Fenster mit einer bestimmten Farbe gefüllt (wenn BGCOLOR beispielsweise gleich (125, 125, 125) ist, ist das Grau).

1.2.4 Der Game-Loop

Das Spiel wird mithilfe einer Endlosschleife verarbeitet. Diese wird auch Game-Loop genannt.

```
while True:
```

```
    pass
```

Innerhalb des Game-Loops findet die Steuerung des Spiels statt. Dazu gibt es das Objekt `pygame.event`, welches mit seiner `get()`-Methode sämtliche Ereignisse (Events) herausgibt, die sich während eines Frames ereignet haben. Es kann als eine Liste gedacht werden, an der an der vorderen Seite Events aufgenommen werden und an der hinteren Seite Events herausgenommen werden. Eine solche Liste wird auch »Queue« genannt.

Events können dann etwa mit `event.type`, `event.key` und `event.pos` ausgewertet werden. Diese geben den Event Typ, die Taste, die gedrückt wurde, sowie die aktuellen Mauskoordinaten an.

1.3 Das Programm

Meine Programme beginnen immer mit einem Kommentarteil, in dem ich den Dateinamen, das Erstellungsdatum, den Titel und den Autor nenne.

```
001 # -*- coding: utf-8 -*-
002 #
003 # schlangenbiss.py
004 #
005 # Created on 27. November 2025, 06:26
006 #
007 # Title: Schlangenbiss
008 # Description: Gameprogram.
```

```
009 # Copyright: (c) 2025 Thomas Kaffka, alle Rechte vorbehalten.  
010 # Author: Thomas Kaffka, Düsseldorf, Germany  
011 #
```

Als erstes werden die nötigen Imports durchgeführt. Das ist `pygame`, `sys` für das Beenden des Programms und `random`, um Zufallszahlen verarbeiten zu können. Danach werden Eventtypen importiert.

```
012 import pygame, sys, random  
013 from pygame.locals import QUIT, KEYDOWN, K_ESCAPE, K_LEFT, K_RIGHT, K_UP,  
    K_DOWN
```

Die folgenden drei Konstanten beziehen sich auf das Format des Spielfensters. Das Spiel wird mit 24 mal 24 Feldern gespielt. Die Höhe und Breite eines Items, welches auf dem Spielfeld abgebildet wird, ist 30 Pixel.

```
014 SCREEN_NDX_X = 24 # index end for the columns.  
015 SCREEN_NDX_Y = 24 # index end for the lines.  
016 FIELD_WIDTH = 30 # pixel height and width of the item pictures.
```

Aus diesen Angaben kann die Pixel-Breite und -Höhe der Oberfläche bzw. des Spielfensters berechnet werden. Für die Ausgabe der Ende-Meldung benötigen wir auch noch die halbe Fensterbreite und -höhe, damit diese zentriert dargestellt werden kann. Dabei muss mit der Funktion `int()` aus der Division ein Integer gemacht werden, damit keine Kommazahl entsteht.

```
017 WINWIDTH = SCREEN_NDX_X * FIELD_WIDTH # width of the game window.  
018 WINHEIGHT = SCREEN_NDX_Y * FIELD_WIDTH # hight of the game window.  
019 HALF_WINWIDTH = int(WINWIDTH / 2)  
020 HALF_WINHEIGHT = int(WINHEIGHT / 2)
```

Für die Ausgabe des Programmnamens und der Programmversion in der Titelzeile des Spielfensters werden zwei Konstanten definiert. Weiterhin wird die Konstante definiert, die die »Frames per Second« steuert.

```
021 PROG_NAME = "Schlangenbiss"  
022 PROG_VERSION = "1.0"  
023 FPS = 30 # frames per second
```

Für die Steuerung der Schlange benötigen wir Richtungskonstanten, weiterhin eine Konstante für die Animation der Schlange. Dann werden noch Konstanten für die verwendeten Farben (Hintergrund sowie Schrift) definiert und der Font, mit dem die Ende-Meldung geschrieben wird. Der kann zurzeit noch keinen Wert haben, da `pygame` noch nicht initialisiert ist. Das geschieht erst in den Zeilen 47 und 48.

```
024 UP = 0
025 DOWN = 1
026 LEFT = 2
027 RIGHT = 3
028 CREEP_ANIMATION_SPEED = 7 # animation speed for creeping.
029 BG_COLOR = (128, 128, 128)
030 GREEN = (0, 255, 0)
031 END_FONT = None # the font for the end screen.
```

In einem Dictionary werden die Items bzw. Bilder, mit denen die Grafik erzeugt wird, gespeichert. Dieses hat auch noch keinen Wert.

```
032 # dictionary
033 IMAGEDICT = None # dictionary for the images.
```

Außerdem werden die Schlüssel für das Dictionary erstellt.

```
034 # item codes
035 BACK = '0000'
036 SNAKE = '0001'
037 APPLE = '0002'
038 OBSTACLE = '0003'
```

Nun werden die globalen Variablen definiert und initialisiert.

```
039 # global variables.
040 creep_count = CREEP_ANIMATION_SPEED + 50 # counter for the creep
    animation.
041 board = [] # the game board
042 snake = [] # the snake segments
043 game_end = False # end of the game
```

Hier beginnt die Hauptfunktion (`main()`) des Programms. Zuerst werden die Variablen und Konstanten, die aus den Reihen der globalen verwendet werden, genannt.

```
044 def main():
045     global creep_count, IMAGEDICT, SOUNDICT, ENDFONT, game_end
```

Dann werden `pygame` sowie der Font für die Ende-Meldung initialisiert. Er hat eine Größe von 48 Pixeln.

```
046     # preparations.
047     pygame.init()
048     ENDFONT = pygame.font.SysFont("Arial", 48)
```

Die Uhr zur Steuerung der Framerate wird geholt.

```
049     fpsClock = pygame.time.Clock()
```

Dann wird die Oberfläche (`SURFACE`) für die Ausgabe der Grafik initialisiert. Danach wird in die Titelleiste des Spielfensters der Programmname und die Version geschrieben sowie das Programmlogo vorweggesetzt.

```
050     SURFACE = pygame.display.set_mode((WINWIDTH, WINHEIGHT), 0, 32)
051     pygame.display.set_caption(PROG_NAME + ' (' + PROG_VERSION + ')')
052     pygame.display.set_icon(pygame.image.load('logo16.jpg'))
```

Nun wird das Dictionary zur Aufnahme der Grafiken für die Items, die im Programm verwendet werden, initialisiert.

```
053     # initialization of the dictionary.
054     IMAGEDICT = {
055         BACK:     pygame.image.load('../pic/Background.png'),
056         SNAKE:    pygame.image.load('../pic/Snake.png'),
057         APPLE:    pygame.image.load('../pic/Apple.png'),
058         OBSTACLE: pygame.image.load('../pic/Obstacle.png')}
```

Jetzt wird das anfängliche Spielfeld gebildet. Dabei werden die Schlange, der Apfel sowie die Hindernisse positioniert. Das siehst du, wenn wir die Funktion `create_board()` besprechen (ab Zeile 96). Die sich dabei ergebende Richtungsvaria-

ble (`direction`) wird zurückgegeben. Das ist dann die anfängliche Kriechrichtung der Schlange, die in den Zeile 82 verwendet wird.

```
059     direction = create_board()
```

Jetzt startet der das Spiel steuernde Game-Loop.

```
060     # game loop.
061     while True:
```

Danach erfolgt die Ereignisverarbeitung. Auf das Ereignis `QUIT` wird mit dem Beenden von `pygame` und dem Beenden des Programmes selbst reagiert. Das Ereignis `QUIT` wird ausgelöst, wenn auf den roten Button oben rechts in der Titelleiste des Fensters geklickt wird. Wird eine Taste betätigt (`KEYDOWN`) und ist es dann die `[ESC]`-Taste, werden auch `pygame` und das Programm beendet.

Bei der Verwendung der Pfeiltasten wird eine bestimmte Richtung gewählt. Dabei darf die Richtung der Schlange nicht einfach umgedreht werden.

Beispiel für die Pfeiltastensteuerung

Wird die `[←]`-Taste betätigt und ist die Richtung, in die sich die Schlange bewegt, nicht »Rechts«, wird als Richtung »Links« verarbeitet. Damit soll verhindert werden, dass der Spieler die Laufrichtung der Schlange einfach umkehrt, da das zum sofortigen Abbruch des Spieles führen würde.

```
062     # key handling.
063     for event in pygame.event.get(): # event handling.
064         if event.type == QUIT:
065             pygame.quit()
066             sys.exit()
067         elif event.type == KEYDOWN:
068             if event.key == K_ESCAPE:
069                 pygame.quit()
070                 sys.exit()
071             elif event.key == K_LEFT and not direction == RIGHT:
072                 direction = LEFT
073             elif event.key == K_RIGHT and not direction == LEFT:
```

```
074         direction = RIGHT
075     elif event.key == K_UP and not direction == DOWN:
076         direction = UP
077     elif event.key == K_DOWN and not direction == UP:
078         direction = DOWN
```

Nun wird das Kriechen der Schlange animiert. Von der Variablen `creep_count` wird bei jedem Schleifendurchlauf eine 1 abgezogen. Ist `creep_count` gleich Null, wird die Variable wieder auf ihren Initialwert gesetzt und danach die Schlange um eine Position bewegt. Das soll bewirken, dass die Schlange nicht zu schnell kriecht.

```
079         creep_count -= 1
080     if creep_count == 0:
081         creep_count = CREEP_ANIMATION_SPEED
082         creep(direction)
```

Wurde während des Spiels das Flag `game_end` gesetzt, wird die Ende-Meldung ausgegeben. Das passiert, wenn sich die Schlange in den Rand, in ein Hindernis oder in sich selbst bewegt (siehe Zeilen 164 bis 173). Die Funktion `end_screen()` bildet zunächst die Ende-Meldung. Deren Ausmaße werden ermittelt, dann wird sie zentriert und schließlich ausgegeben.

```
083     if game_end:
084         mapEnd = end_screen()
085         mapSurfRect = mapEnd.get_rect()
086         mapSurfRect.center = (HALF_WINWIDTH, HALF_WINHEIGHT)
087         SURFACE.blit(mapEnd, mapSurfRect)
```

Im anderen Fall wird das Spielfeld erstellt (`print_board()`), dessen Ausmaße ermittelt und dann ausgegeben.

```
088     else:
089         mapBoard = print_board()
090         mapBoardRect = mapBoard.get_rect()
091         SURFACE.blit(mapBoard, mapBoardRect)
```

Jetzt wird das Spielfenster upgedatet, also die Änderungen sichtbar gemacht, und danach so lange gewartet, dass sich 30 »Frames per Second« ergeben. Das hängt

von der jeweiligen Prozessorleistung deines Computers ab, auf dem das Spiel verarbeitet wird.

```
092     pygame.display.update()
093     fpsClock.tick(FPS)
```

Nun kommen wir zu den Funktionen des Programms. Die erste Funktion erstellt das Spielfeld. Dazu wird in zwei geschachtelten `for`-Schleifen in einer Linienliste der Schlüssel `BACK` gespeichert und die Linienlisten der Spielfeldliste (`board`) angefügt. Das Ergebnis ist eine geschachtelte Liste. In der Liste `board` befinden sich Listen mit den Spielfeldzeilen.

```
094 #####
095 # creates the initial board.
096 def create_board():
097     global board, snake
098     # make the background.
099     for y in range(SCREEN_NDX_Y):
100         line = []
101         for x in range(SCREEN_NDX_X):
102             line.append(BACK)
103         board.append(line)
```

Das sieht dann so aus:

```
[['0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000'], ['0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000'], ['0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000',
'0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000', '0000'],
['0000', ...
```

Die Liste `board` ist eine Liste, bestehend aus Listen. Die einzelnen Elemente können ganz einfach mit `board[y][x]` adressiert werden.

Als nächstes werden drei Segmente der Schlange erzeugt. Für das erste Segment werden Zeilen- und Spaltenkoordinaten zufällig ermittelt. Diese aber jeweils drei

Stellen vom Rand entfernt, da die Schlange ja später zum Rand hin gebaut werden kann. In der Liste `sanke` werden sämtliche Koordinaten der Schlange gespeichert. Das versetzt uns in die Lage, beim Kriechen jeweils das Ende der Schlange löschen zu können (siehe Zeilen 177 bis 179). Danach wird, für das spätere Ausgeben des Spielfeldes, der Schlüssel der Schlange in das Spielfeld eingetragen.

Dann wird die Richtung (`direction`) bestimmt, in die die Schlange gebaut werden soll. Beachte, dass die Grenze für die Zufallszahl bei der ermittelten Zufallszahl nicht berücksichtigt wird. Es werden also Zufallszahlen zwischen Null und 3 ermittelt.

Abhängig von der Richtung wird die Schlange gebaut (`create_snake()`). Und dann wird deren Bewegungsrichtung umgekehrt, da die Schlange im ersten Falle nach oben gebaut wird und sich dann nach unten bewegen muss. In den Parametern `y` und `x` werden die Koordinaten des Schlangenkopfs übergeben. Die beiden anderen Parameter sind für die Baurichtung der Schlange zuständig. Das siehst du später bei der Beschreibung Funktion `create_snake()`.

```
104     # make the snake.
105     y = random.randrange(3, SCREEN_NDX_Y - 3)
106     x = random.randrange(3, SCREEN_NDX_X - 3)
107     snake.append([y, x])
108     board[y][x] = SNAKE
109     direction = random.randrange(4)
110     if direction == UP:
111         create_snake(y, x, -1, 0)
112         direction = DOWN
113     elif direction == DOWN:
114         create_snake(y, x, +1, 0)
115         direction = UP
116     elif direction == LEFT:
117         create_snake(y, x, 0, -1)
118         direction = RIGHT
119     elif direction == RIGHT:
120         create_snake(y, x, 0, +1)
121         direction = LEFT
```

Die Funktion zur Erzeugung des Apfels wird aufgerufen. Da ein Apfel nach dem Essen ja wieder neu generiert werden muss, habe ich dafür eine eigene Funktion vorgesehen, die von zwei Stellen aus aufgerufen wird (Zeilen 122 und 167).

```
122     create_apple()
```

Als letztes werden die Hindernisse erstellt. Es werden bis zu 20 Hindernisse gebaut. Deren Koordinaten werden jeweils ermittelt, und wenn sich auf dem Spielfeld weder ein Apfel noch eine Schlange befindet, wird ein Hindernis auf das Spielfeld gesetzt.

```
123     # create obstacles.
124     for ndx in range(random.randrange(20)):
125         y = random.randrange(SCREEN_NDX_Y)
126         x = random.randrange(SCREEN_NDX_X)
127         if board[y][x] == BACK:
128             board[y][x] = OBSTACLE
```

Um die Richtung in der Zeile 59 verwenden zu können, wird sie hier zurückgegeben.

```
129     return direction
```

Die Funktion, die die Schlange baut, geht so vor. In den Variablen `y` und `x` werden, wie bereits beschrieben, die Koordinaten des Schlangenkopfs übergeben. In den Variablen `yy` und `xx` steht die jeweilige Bauvorschrift.

Soll die Schlange nach oben gebaut werden, steht in `yy` eine -1 und in `xx` eine Null. Dann werden auf die Koordinaten des Kopfes beide Werte addiert. Dabei ändert sich nur die Zeile, da in `xx` ja eine Null steht. In der Liste `board` wird das zweite Segment der Schlange gespeichert und in der Liste `snake` die Koordinate. So wird auch mit dem dritten Segment verfahren.

```
130     #####
131     # creates the snake.
132     def create_snake(y, x, yy, xx):
133         global board, snake
134         x += xx
135         y += yy
```

```
136     board[y][x] = SNAKE
137     snake.append([y, x])
138     x += xx
139     y += yy
140     board[y][x] = SNAKE
141     snake.append([y, x])
```

Der Apfel muss auf einem leeren Feld platziert werden. Dazu wird zunächst eine zufällige Koordinate ermittelt. Dann wird dies in einer `while`-Schleife so lange fortgeführt, bis die Position auf dem Spielfeld leer ist. Dort wird dann der Apfel hingesetzt.

```
142 #####
143 # creates an apple.
144 def create_apple():
145     y = random.randrange(SCREEN_NDX_Y)
146     x = random.randrange(SCREEN_NDX_X)
147     while board[y][x] != BACK:
148         y = random.randrange(SCREEN_NDX_Y)
149         x = random.randrange(SCREEN_NDX_X)
150     board[y][x] = APPLE
```

Die folgende Funktion bewegt die Schlange. Es wird die Koordinate des Kopfes der Schlange (`snake[0]`) geholt. Abhängig von der Richtung werden nun diese Koordinaten verändert. Ist die Schlange dadurch in den Rand gelaufen, ist das Spiel zu Ende. Wenn sie auf einen Apfel gekrochen ist, wird ein neuer Apfel erzeugt. Dann wird am vorderen Ende der Schlange die aktuelle Koordinate angefügt und ein entsprechender Schlüssel wird auf dem Spielfeld gespeichert. Die Schlange ist dadurch gewachsen, da das Löschen des Endes der Schlange nicht verarbeitet wird.

Kriecht die Schlange in sich selbst oder in ein Hindernis, ist das Spiel jeweils zu Ende. Der abschließende `else`-Zweig bezieht sich auf das Kriechen der Schlange. An das vordere Ende der Schlange (Index Null) wird die aktuelle Koordinate angefügt. Im Spielfeld wird der Schlüssel dieser Koordinate der Schlange gespeichert. Dann wird die letzte Koordinate der Schlange geholt, das Spielfeld mit `BACK` gelöscht und die letzte Koordinate in der Liste `snake` mit `pop()` gelöscht.

Stichwortverzeichnis

Symbole

2D-Spiele 16

__main__ 29

__name__ 29

ϵ -greedy Strategie 172

A

Abbildungen 36, 214

Abbruchbedingung 251

abs() 263

Agent 170

Aktion 170, 178, 179, 187, 201, 202, 203, 206

Aktuelle Belohnung 171

Algorithmus 171, 172

Alpha 202

Alphakanal 115

and 253

Animation 33, 39, 180, 197

Animationsbilder 47

Animationsindex 39

Antialiasing 28

Attribut 69, 81, 97, 125, 194, 279

Ausgabebereich 68

Avatar 103, 105, 108, 114, 117, 122, 123, 125, 168, 175, 182, 185, 190

Axon 139

B

Bellmann-Gleichung 205

Belohnung 168, 203, 206

Beschriftung 94

Beste Aktion 204

Bestrafung 168, 203, 206

BGCOLOR 37

Bild 173, 182, 192

Binär 262

Binärzahlen 174

Blätter 212

break 261

Buchstabenfont 34

Button 184, 192

C

Clock 218

Compiler 245

csv-Datei 159

D

Datei 88, 96, 104, 120

Datenerfassung 158

Datenerhebung 162

def 272

Dendrit 139

Dezimalzahlen 261

Dictionary 19, 28, 33, 36, 71, 73, 115, 144, 202, 266, 271

Differenzmenge 271

Directory 63, 118, 120

Diskontierungsfaktor 171

Dynamische Programmierung 168

E

Elemente 23

elif 42

else 255

Elternklasse 81, 97, 172, 194, 213, 230

Ende Meldung 28
 Ende Message 157
 Ende-Text 95
 Endlosschleife 17, 254, 261
 Entscheidung 170
 Episode 170, 178, 184, 201
 Episodenfolge 178
 Episodengrenze 201
 Epsilon 170, 175, 202
 Eratosthenes 268
 Ereignis 21, 37, 145
 Ereignisverarbeitung 21
 Erwartete Belohnung 171
 Event 17, 145
 Event-Queue 55, 111, 126, 128, 216
 Eventtypen 18
 Explosionsgeräusch 38
 Exponentialfunktion 263

F

Fakultät 264
 Falsch 252
 False 252
 Framerate 20
 Farbe 17, 34, 37, 97, 106
 Feedback 170
 Fehlermeldungen 29
 Feldkoordinaten 186, 189
 Fenster 14
 Fensterinhalt 224
 Fibonacci-Zahlen 253
 Flag 22, 38, 40, 86, 107, 131, 143, 175, 188, 215, 216, 223, 231, 246
 Font 16, 34, 59, 88, 95, 106, 111, 113, 175, 181, 215, 218
 for-Schleife 250
 for-Statement 250
 FPS 11, 17, 33, 39, 144, 217
 Frame 17, 37, 87, 153
 Framerate 16, 33, 39

Frames per Second 12, 17, 18, 22, 54, 59, 83, 106, 141, 144, 149, 173, 175, 181
 Frames pro Sekunde 41
 Funktion 25, 26, 265, 272

G

Game-Loop 11, 16, 17, 21, 35, 37, 41, 48, 55, 84, 107, 109, 110, 145, 149, 176, 216
 Game Over 86
 Gamma 202
 Gemischt 56
 Gesichtserkennung 138
 getoggelt 147
 Getter 75, 231
 Gewicht 140
 Gleich-Bedingung 252
 Gleitkommadarstellung 261
 global 273
 globale Attribute 105, 118
 globale Variable 19, 54, 83, 105, 111, 143, 175
 Grafik 19, 57
 Güte des Netzes 161

H

Hauptfunktion 20, 84, 143, 216
 Hauptprogramm 50, 53, 69, 107, 113, 158, 172, 194, 230
 Haupt-Programmfunktion 35
 Hexadezimal 262
 Hiddenschicht 140, 160
 Highscore 50, 84
 Hintergrund 107, 150
 Hintergrundfarbe 54, 63, 68, 99, 112, 114, 145, 175, 192, 215, 225, 227
 Hintergrundmusik 87
 Hinweis 58, 226
 Hinweistext 68
 Hyperparameter 202

I

Icon 115
 IDE 10, 206
 if-Statement 255
 Import 18, 32, 54, 105, 141, 213
 Index 113, 266
 Info-Bereich 114
 Informationstext 114
 Initialisierung 55, 84, 107, 111, 216, 217
 Initialisierungsmethode 279
 Init-Methode 101, 124, 125, 133, 194,
 196, 230, 237
 Inlinekommentar 33, 35
 Inputschicht 140, 160
 instanziiert 90
 Integrierte Entwicklungsumgebung 10
 Interpreter 245
 Item 28, 40, 43, 48
 Iterative Lösung 264

K

Kategorie 138, 140
 KI 158, 168
 Klasse 53, 62, 69, 75, 80, 81, 90, 97, 101,
 105, 118, 120, 125, 132, 172, 196, 213,
 230, 237, 238, 279
 Klick 184
 Knoten 212
 Kollision 149
 Kommazahlen 248
 Kommentar 17, 69, 246
 Kommentarzeile 277
 Konsole 97, 206
 Konstante 54, 105, 142, 174, 213, 246
 Koordinaten 14, 195
 Koordinatendifferenz 162
 Koordinatenumwandlung 43
 Künstlichen Intelligenz 138

L

Lernen 168
 Lernen durch Verstärkung 168
 Lernphase 169, 178
 Lernrate 171
 Level 84, 103, 106, 107, 114, 120, 122
 Leveldatei 105
 Levelmanager 106, 108, 120
 list 259
 Liste 23, 36, 41, 60, 62, 63, 73, 123, 131,
 147, 151, 219, 266
 Logische Operatoren 253
 Lokale Variable 107

M

Mahjongg 51
 main() 35
 math 32, 263, 272
 Maus 37
 Mausbutton 57
 Mausklick 38, 177
 Mauskoordinaten 17, 217
 max() 263
 Message 181, 193, 215, 216, 218, 222,
 225, 228
 Messagezeile 170
 Methode 64, 75, 77, 81, 99, 100, 110, 122,
 126, 133, 194, 279
 min() 263
 Mini-Max-Algorithmus 212
 Mini-Max-Strategie 239
 Mischen 62
 Mixer 59
 Mögliche Aktion 204

N

Nervenzelle 138
 Netz 139
 Neuronales Netz 137, 138, 159, 160, 164
 Nimm-Spiel 274

None 33, 83, 128, 246

Normierung 160

not 253

O

Oberfläche 16, 20, 27, 28, 35, 40, 46, 49,
59, 83, 86, 88, 106, 111, 181, 215, 218

Objekt 54, 59, 110, 115, 188, 215, 220,
279

Objektorientierte Programmierung 278

oktal 262

OneHotEncoder 164

Optimaler Pfad 208

Optimaler Weg 169

Optimale Strategie 170

or 253

Outputschicht 140, 160

P

Palindrom 260

Parameter 64, 100, 126

pass 246

Pfad 162, 179

pip 10

Pixel 14, 118

Pixelkoordinaten 43, 61, 74, 186, 195,
197

Primzahlen 268

Printstatement 247

Programmfenster 14, 59

Programmiersprache 9, 245

Programmlogo 20

Programmname 20

Programmschleife 249

Punkte 40, 49, 84, 143, 157, 175, 179,
203, 229, 233, 241, 243

pygame 10, 16

Python 9, 245

Python-Referenz 248

Q

Q-Learning 168, 171, 178, 182

Q-Learning-Algorithmus 172

q-Tabelle 171, 203

Queue 17

QUIT 38

R

random 32

range 251

Rekursion 44, 243

Rekursionsstufe 242

Rekursionstiefe 213, 238

Rekursiv 44

Rekursive Lösung 264

Retro-Spiele 16

return 265, 273

RGB-Farben 15

Richtungskoordinaten 199

round() 263

S

Scaler 164

Schatteneffekt 95

Schlangenbiss 13

Schlüssel 19, 28, 34

Schnittmenge 271

Schriftart 83

self 279

set 259

Set 266, 270

Setter 75, 231

Sicherheit 140

Situation 201, 203, 204

Situationstabelle 203

Skript 11

Sound 36, 59, 70, 73, 126, 128, 173, 182,
186, 190, 214, 222

Sounddirectory 119

Spielbaum 212

Spielfeldkoordinaten 217, 219, 221

Spielstein 52

Spielsteine mischen 54

Spyder 10, 29, 247

Startbildschirm 108
 Statement 122, 155, 245
 Strategie 170, 220, 221, 238, 239, 242
 Strategiealternative 227
 String 114, 157, 163, 247, 256
 super() 196, 237
 Synapse 139
 Syntax 104
 Syntaxfehler 246
 sys 32

T

Tastatur 37
 Terminal 247
 Text 94, 96, 109, 113, 157
 Textfarbe 28, 113
 Texthintergrund 28
 Textobjekt 28, 49, 68, 113, 114, 157
 Textposition 113
 Textschatten 111
 Time Clock 35
 Training 140, 158, 205
 Trainingsdaten 138
 Trainingsprozess 161
 Transparent 47, 116, 190
 Treffersound 154
 True 252
 Tupel 15, 61, 125, 133, 266, 270

U

Umwelt 170

V

Variable 248
 Vereinigungsmenge 271
 Versuch und Irrtum 168

W

Wahr 252
 Wahrheitswerte 252
 while-Schleife 251
 while-Statement 251
 Wurzel 212
 Wurzelfunktion 263

X

XML-Datenstruktur 166
 XOR 271

Z

Zahlen 261
 Zeichenkette 247, 256
 Zeilenkoordinate 14
 Zufällige Aktion 204
 Zufällige Koordinate 26
 Zufallszahl 18, 24, 41, 137, 152, 277
 Zustand 170