2024

IT Education & Consulting

Raphael Kawasch

Kawasch ITeduCon

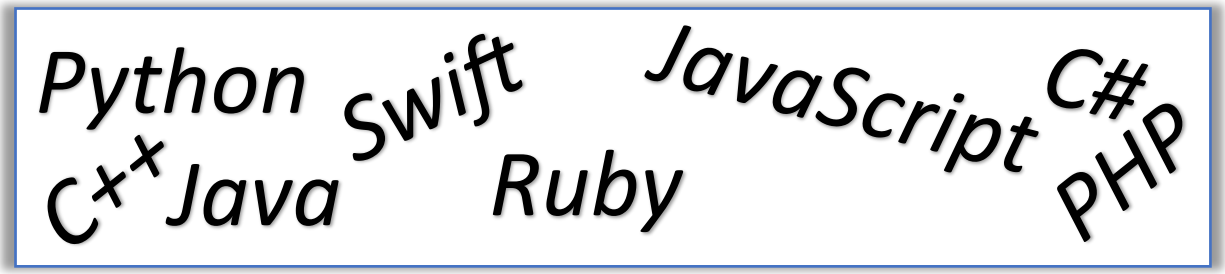
30.11.2024

Einführung

Was ist Python?

„Computer erledigen ihre Aufgaben nach schrittweisen Anweisungen. Die Programme – der „Code“ – werden in verschiedenen Programmiersprachen geschrieben und Python wird dafür weltweit mit am häufigsten verwendet.“ (Vorderman, 2019)

Welche Programmiersprachen kennst du noch?



Warum Python?

Python ist leicht zu lernen, da man sich um einige Dinge nicht kümmern muss, die in anderen Programmiersprachen erforderlich sind. Zudem ist Python plattformunabhängig, aber dennoch sehr leistungsfähig. Für schulische Zwecke eignet sich Python besonders gut, da hier auch viele Beispiele existieren (Bücker, Videos, Tutorials) und die Sprache auch im Beruf und in der Wissenschaft Anwendung findet. Auch zum Entwickeln von Spielen ist Python eine ideale Sprache 😉

Python für Spielemacher

„Videospiele sind Computerprogramme mit einer Reihe von Anweisungen. Mit Python lassen sich viele verschiedene Arten von Spielen programmieren – da ist sicherlich für jeden Gamer was dabei.“ (Vorderman, 2019)

Python installieren

Bevor wir loslegen können müssen wir unsere Arbeitsumgebung vorbereiten. Dazu installieren wir Python auf unserem Computer. Von der offiziellen Webseite können wir uns Python herunterladen https://www.python.org/. Unter Windows erhalten eine python*.exe (wobei * ein als Platzhalter für die Version anzusehen ist), diese Datei müssen wir installieren. Wenn ihr nicht mehr wisst wie man ein Programm installiert, fragt bitte eure Eltern um Hilfe. Mit Python wird auch die Entwicklungsumgebung IDLE installiert. Damit haben wir bereits die Basis für unser erstes Spiel.
In meinem VHS-Kurs verwenden wir eine online zur Verfügung stehende Alternative von IDLE, ihr könnt sie hier aufrufen https://www.online-python.com/.

Pygame Zero

Für die Spieleentwicklung mit Python gibt es einen Programmbaustein (Modul) **Pygame Zero** der uns viele komplizierte Programmierungen abnimmt. Damit können wir uns auf unsere Spieleidee konzentrieren und schneller unser erstes Spiel programmieren. Für die fortgeschrittenen Spieleentwickler unter euch, kann ich auch den Programmbaustein Pygame empfehlen, hier werden euch nicht ganz so viele Dinge abgenommen, dafür müsst ihr dann aber auch mehr Details programmieren.

Um das Modul zu installieren öffnen wir eine Eingabeaufforderung und setzen folgenden Befehl ab:

```
> python -m pip install -U pip
```

Damit installieren wir Paketverwaltung für Python.

Anschließend installieren wir die benötigten Module.

```
> pip install pygame
> pip install pgzero
```

Jetzt kanns los gehen, wir programmieren unser erstes Spiel 😊

ABER wie geht das?

Dazu müssen wir erst einmal die Sprache Python und ihre Bestandteile kennenlernen. Los geht's …

Grundlagen der Sprache

Variablen

„Variablen speichern und benennen einzelne Informationen. Sie sind sehr vielseitig einsetzbar, denn sie zeichnen zum Beispiel Punktzahlen oder die Zahl deiner verbleibenden Leben auf." (Vorderman, 2019)

Eine Variable ist wie eine Box in der wir Dinge ablegen können, wie erwähnt z.B. die Anzahl unserer noch verbleibenden Leben. Wir können dort aber auch den Namen unseres Helden oder die Farbe unserer Haare ablegen. In Python müssen wir uns keine Gedanken 🤔 was wir in unserer Variable (Box) ablegen wollen. Das Ablegen von Dingen in unserer Variable nennt man Zuweisung. Lasst uns nun Variablen anlegen und Informationen zuweisen.

```
#Variable für die Punkte anlegen und initial auf 0 setzen
punkte = 0
```

Damit wir nun auch testen können was in unserer Variablen gespeichert ist, rufen wir eine Funktion auf, die den Inhalt der Variablen am Bildschirm ausdruckt. Was eine Funktion ist, sehen wir uns später noch genauer an.

```
#Variable für die Punkte anlegen und initial auf 0 setzen,
anschließend ausgeben
punkte = 0
print(punkte)
```

Nach dem wir nun gesehen haben, wie Variablen funktionieren, können wir damit auch kleine Programme erstellen. Lasst uns mal etwas rechnen!

```
zahlA = 2
zahlB = 3
ergebnis = zahlA + zahlB
print(ergebnis)
```

Ihr seid dran, probiert verschiedene Operatoren Plus, Minus, Mal, Geteilt (+,-,*,/).

Das Ganze geht nun auch mit **Zeichenketten** (Strings).

```
name = "Hugo"
print(name)
```

Zeichenketten kann man mittels **+** Operator auch verbinden.

```
name = "Hugo"
begruessung = "Hallo "
nachricht = begruessung + name
print(nachricht)
```

Die Länge einer Zeichenkette können wir mit der Funktion len() abfragen.

```
name = "Hugo"
begruessung = "Hallo "
nachricht = begruessung + name
print(nachricht)

print(len(nachricht))
```

In einem Spiel hat man häufig viele Dinger einer gleichen Sache, z.B. Spielkarten, dazu kann man entweder jeder Karte einzeln definieren oder eine Liste mit allen Karten erzeugen.

```
#Anlegen aller Karten einzeln

karte1 = "1 herz"
karte2 = "2 herz"
karte3 = "3 herz"
karte4 = "4 herz"
karte5 = "5 herz"
karte6 = "6 herz"
```

```
# Anlegen aller Karten in einer Liste
karten = ["1 herz", "2 herz", "3 herz", "4 herz", "5 herz", "6
herz"]
```

Wie ihr seht, ist die Liste eine viel kompaktere Möglichkeit viele gleiche Dinge anzulegen und zu verwalten.

Um eine Karte aus der Liste anzusprechen um sie z.B. zu ziehen benennt man die Position der Karte in der Liste.

```
# Anlegen aller Karten in einer Liste
karten = ["1 herz", "2 herz", "3 herz", "4 herz", "5 herz", "6 herz"]


#Erste Karte in der Liste
print(karten[0])
#>>> "1 herz"
#Letzte Karte in der Liste
print(karten[5])
#>>> "6 herz"
```

Entscheidungen

In einem Spiel muss man auch an vielen Stellen eine Entscheidung treffen; nehmen wir an, wir schießen einen Elfmeter, der Spieler muss entscheiden ob er nach links oder nach rechts schießen möchte, oder gar in die Mitte des Tores. Die Entscheidung hängt davon ab, an welcher Stelle der Torwart steht oder in welche Richtung er lieber springt. In einem Programm muss man diese Fälle einzeln vorsehen.

Dazu helfen uns Wahrheitswerte sogenannte Boolsche Werte (True / False).

Einen Boolschen Wert bekommen wir zurück, wenn wir logische Operatinen, wie z.B. einen Vergleich auf Gleichheit (==) durchführen.

Achtung: Das einfache = benutzen wir für eine Zuweisung; Das doppelte == verwenden wir zum Vergleichen!
Hier eine Liste der Logischen Operatoren:

Logische Operatoren (Symbol)	Bedeutung
==	ist gleich
!=	ist nicht gleich
<	kleiner als
>	größer als
<=	kleiner gleich
>=	größer gleich

Am Beispiel des Elfmeterschießens könnte das nun so aussehen:

```
position = "rechts"
if position == "rechts":
      schuss = "links"
elif position == "links":
      schuss = "rechts"
else:
      schuss = "mitte"
print(schuss)
```

Wir können auch Vergleichen ob etwas größer oder kleiner ist. Lasst uns mal prüfen ob wir noch mindestens ein Leben (> 0) haben.

```
leben = 5
print(leben > 0)
```

Schleifen

„In einem Spiel gibt es immer wieder Codeabschnitte, die mehrmals ausgeführt werden müssen. Wenn du sie immer wieder schreiben müsstest, wäre das sehr ermüdend. Glücklicherweise gibt es die Schleifen – in zahlreichen unterschiedlichen Formen.“ (Vorderman, 2019)

Zählschleifen können uns dabei helfen, eine Aufgabe eine bestimmte Anzahl oft zu wiederholen.

```
for count in range (1,11):
    print("Programmieren macht Spaß")
```

Das funktioniert nun auch für das ziehen einer Karte.

```
karten = ["1 herz", "2 herz", "3 herz", "4 herz", "5 herz", "6 herz"]
for karte in karten:
    print("Meine Karte ist: " + karte)
```

Und auch komplexere Aufgaben wie das Kombinieren von zwei Listen ist möglich:

```
karteNummer = ["1","2","3","4"]
karteFarbe = ["herz","karo","pike","raute"]
index = 0
for each in karteNummer:
    print("Deine Karte ist: " + karteNummer[index] + " " + karteFarbe[index])
    index = index +1
```

Solltest du einmal nicht genau wissen, wie oft etwas wiederholt werden muss, kann man dazu die while Schleife anwenden. Der Code in der Schleife wird solange ausgeführt, bis eine bestimmte Bedingung (Abbruchbedingung) eintritt. Achtung: wenn ihr die Abbruchbedingung vergesst, wird der Code unendlich oft ausgeführt.

Ein Beispiel für eine while Schleife die nie endet:

```
while True:
    print("Das ist eine Schleife die nie endet!")
```

Und hier noch ein Beispiel wie man eine solche Schleife auch vernünftig anwenden kann:

```
antwort = input("Wasser-Ball werfen? (j/n)")
while antwort == "j":
    print("Platsch!!!")
    antwort = input("Noch einen Wasser-Ball werfen? (j/n)")
print("Tschüss!")
# (Vorderman, 2019) Seite 39
```

Konstruiere eigene Beispiele und schau was passiert 😉

Funktionen

„Funktionen sind sehr nützlich, weil sie viel Schreibarbeit ersparen. Sie sind benannte Codeblöcke, die du immer wieder verwenden kannst, indem du einfach nur ihren Namen schreibst! Python hat bereits einige vordefiniert Funktionen, aber du kannst auch jederzeit eigene Funktionen schreiben, die genau zu den Aufgaben in deinem Spiel passen.“ (Vorderman, 2019)

Eine vordefinierte Funktion haben wir schon kennengelernt, die print Funktion

```
print("Das ist ein Parameter")

# (Vorderman, 2019) Seite 40
```

Funktionen können aber auch anders aufgerufen werden.

```
# "Funktionen sind in ".count("in ")
print("Funktionen sind in ".count("in "))
#"blau ".upper()
print("blau ".upper())
# (Vorderman, 2019) Seite
```

Natürlich kannst du auch eigene Funktionen definieren, das geht so:

```
def fruit_score():
    print(10)
fruit_score()

# (Vorderman, 2019) Seite 42
```

Und jetzt etwas komplizierter.

```
def fruit_score(fruit):
    if fruit == "Apfel":
        print (10)
    elif fruit == "Orange":
        print(5)
fruit_score("Apfel")
fruit_score("Orange")

# (Vorderman, 2019) Seite 42
```

Deine Funktion muss manchmal auch etwas ausführen und dann einen Wert an das Programm zurückgeben. Dazu verwenden wir **return**. Hier mal ein Beispiel:

```
def fruit_score(fruit):
      if fruit == "Apfel":
            return 10
      elif fruit == "Orange":
            return 5
#Den Rückgabewert verwenden

apple_score = fruit_score("Apfel")
orange_score = fruit_score("Orange")

total = apple_score + orange_score
print(total)

# (Vorderman, 2019) Seite 43
```

Spiele Programmieren

Nachdem wir nun die Grundlagen gelernt haben, können wir endlich loslegen unser erstes Spiel zu programmieren. Jeder gute Programmierer startet dabei aber nicht direkt mit dem Coding seines Spiels. Zunächst wird das Vorhaben geplant.

UML-Diagramm

Für die anschauliche Planung und Dokumentation deines ersten Spiels erstellen wir ein UML-Diagramm. Ein UML (Unified Modeling Language) Diagramm ist eine grafische Repräsentation deines Programmcodes, es stellt sicher, dass du bei der Programmierung nichts vergisst und deinen Code nicht wieder verändern musst, da wichtige Abläufe vergessen wurden.

Ein UML-Diagramm kannst du auf Papier zeichnen oder ein online Tool verwenden. Im Kurs verwenden wir den online Editor von draw.io, der sehr häufig zu Einsatz kommt, auch später im Berufsleben 😉

Einfacher UML-Editor: https://www.umletino.com/umletino.html

Sehr beliebter grafischer draw.io Editor: https://app.diagrams.net/

Die einzelnen Elemente eines UML-Ablaufdiagramms hast du im Kurs kennengelernt. Solltest du diese vergessen, kannst du z.B. hier nochmal nachlesen:

https://www.cybermedian.com/de/a-comprehensive-guide-to-14-types-of-uml-diagram/

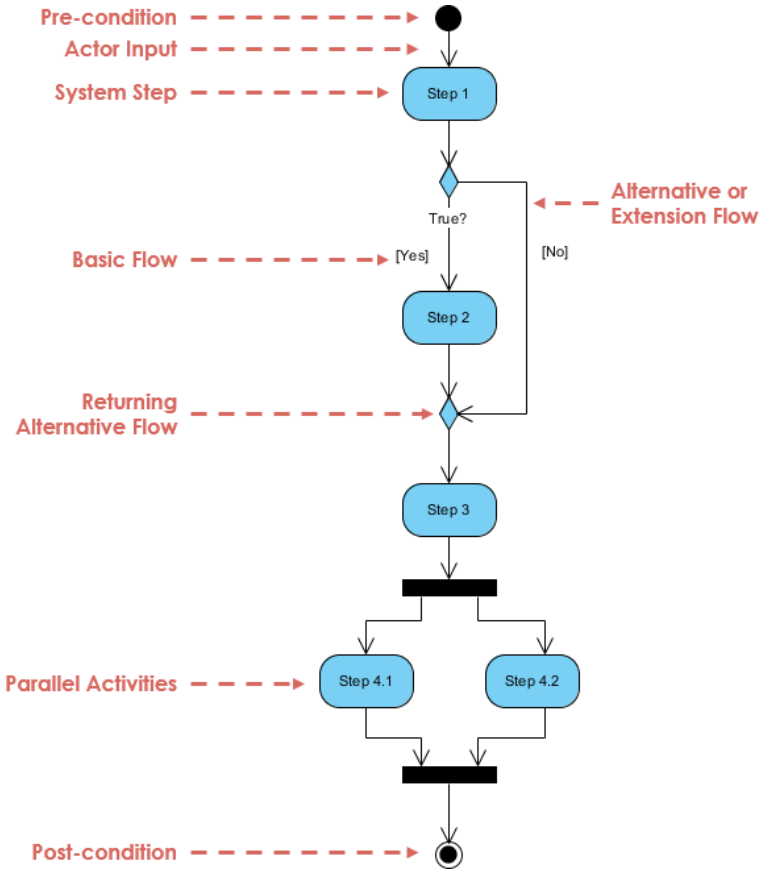


Abbildung 1 Elemente des Aktivitätsdiagramms von https://www.cybermedian.com/de/a-comprehensive-guide-to-14-types-of-uml-diagram/

Planen wir also unser Spiel

Titel dies Spiels:

Obstgarten

Ziel des Spiels:

Möglichst viel Obst durch Anklicken einsammeln

Spiel Ablauf:

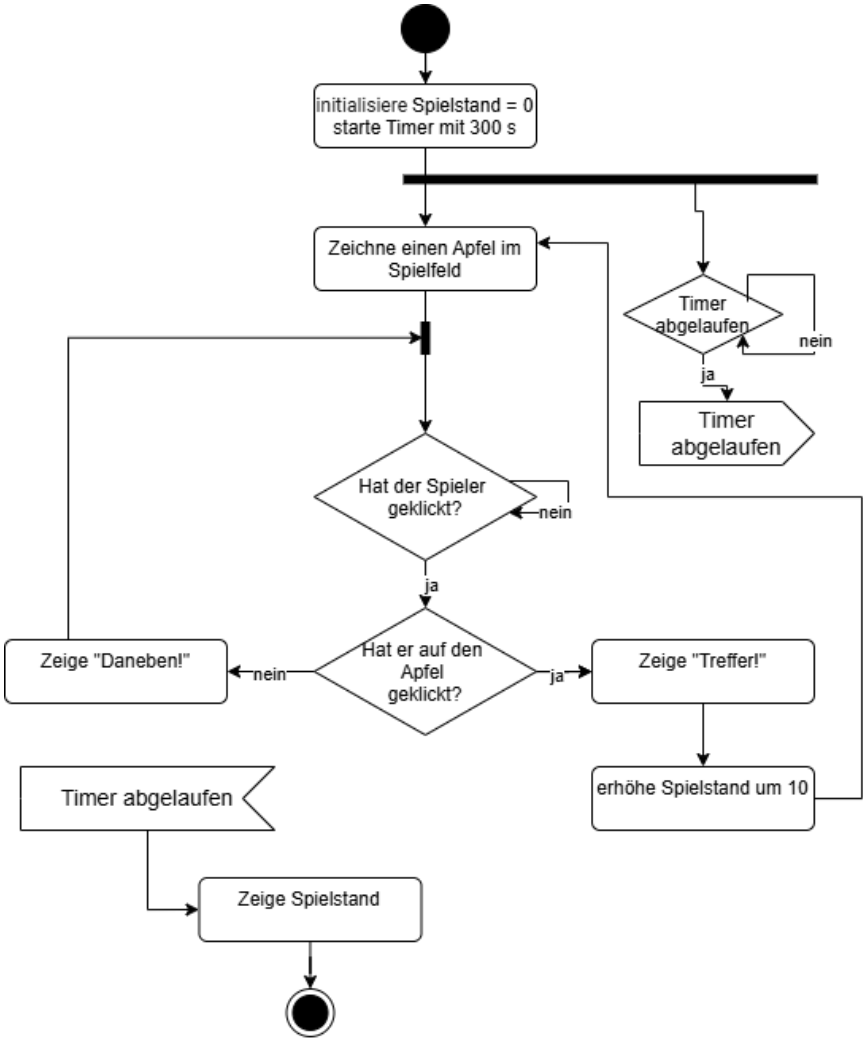
Der Spieler muss in 300 Sekunden möglichst viel Obst einsammeln. Nach Ablauf der 300 Sekunden wird das Spiel beendet und der Spieler sieht seine erreichte Punktezahl am Bildschirm. Jeder gesammelte Apfel ergibt 10 Punkte.

Erweiterungsmöglichkeiten:

Sollte der Spieler das Obst verfehlen, werden Punkte abgezogen.

Mehr Obstsorten mit verschiedenen Punktzahlen implementieren.

UML-Grafik zum Spiel:



Nachdem wir das Spiel geplant haben können wir nun loslegen und unseren Code schreiben.

Damit unser Programm später auch funktioniert, müssen wir zunächst das Modul pgzero einbinden und starten, im Beispiel geben wir das Wort „Hallo“ am Spielfeld aus.

```
#Modul einbinden
import pgzrun
#Funktion definieren
def draw ():
    screen.draw.text("Hallo", topleft = (10, 10))
#Modul starten
pgzrun.go()
```

Das Einbinden und Starten des Modules von pgzero ist für jedes Programm notwendig. Die definierte Funktion bzw. der Programmablauf kann variieren.

Wir öffnen dazu IDLE, schreiben unser Programm und sichern den Code als Grundlage für unser Spiel als Obst.py auf unserem Laufwerk.

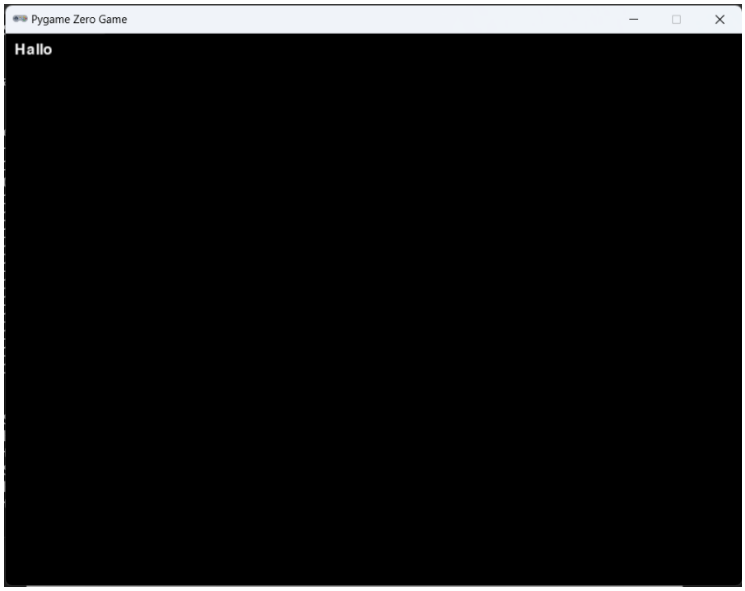
Nun können wir eine Eingabeaufforderung öffnen und in unser Laufwerk wechseln.

```
cd <Pfad zu unserem Laufwerk>
```

Anschließend können wir das Spiel starten.

```
python obst.py
```

Unser Programm startet und wir sehen Hallo im Spielfeld stehen.



Nun machen wir uns an die einzelnen Bestandteile unseres Spieles. Als erstes erzeugen wir unser Spielfeld.

```
import pgzrun
WITH = 600
HEIGHT = 600
def draw():
    screen.clear()
pgzrun.go()
```

Nun brauchen wir noch einen Punktestand, den wollen wir mal oben links in der Ecke anzeigen, ich schlage vor wir drucken ihn in grüner Farbe.

```
import pgzrun
WITH = 600
HEIGHT = 600
#Punktestand erstellen
SCORE = 0
def draw():
    screen.clear()
    #Punktestand anzeigen
    screen.draw.text("Punkte: " + str(SCORE), color="blue",
                     topleft=(10 ,10))
pgzrun.go()
```

Nun kümmern wir uns um unseren Timer, dazu müssen wir einiges beachten. Wir müssen erst einmal den Timer starten, dann müssen wir prüfen ob die Zeit schon abgelaufen ist und wenn ja das Spiel beenden und den Punktestand anzeigen.

```
import pgzrun
WITH = 600
HEIGHT = 600
#Punktestand erstellen
SCORE = 0
#Variable zum Prüfen ob der Timer abgelaufen ist
GAME_OVER = False
#Timer Funktion
def time_up():
    global GAME_OVER
    GAME_OVER = True
```

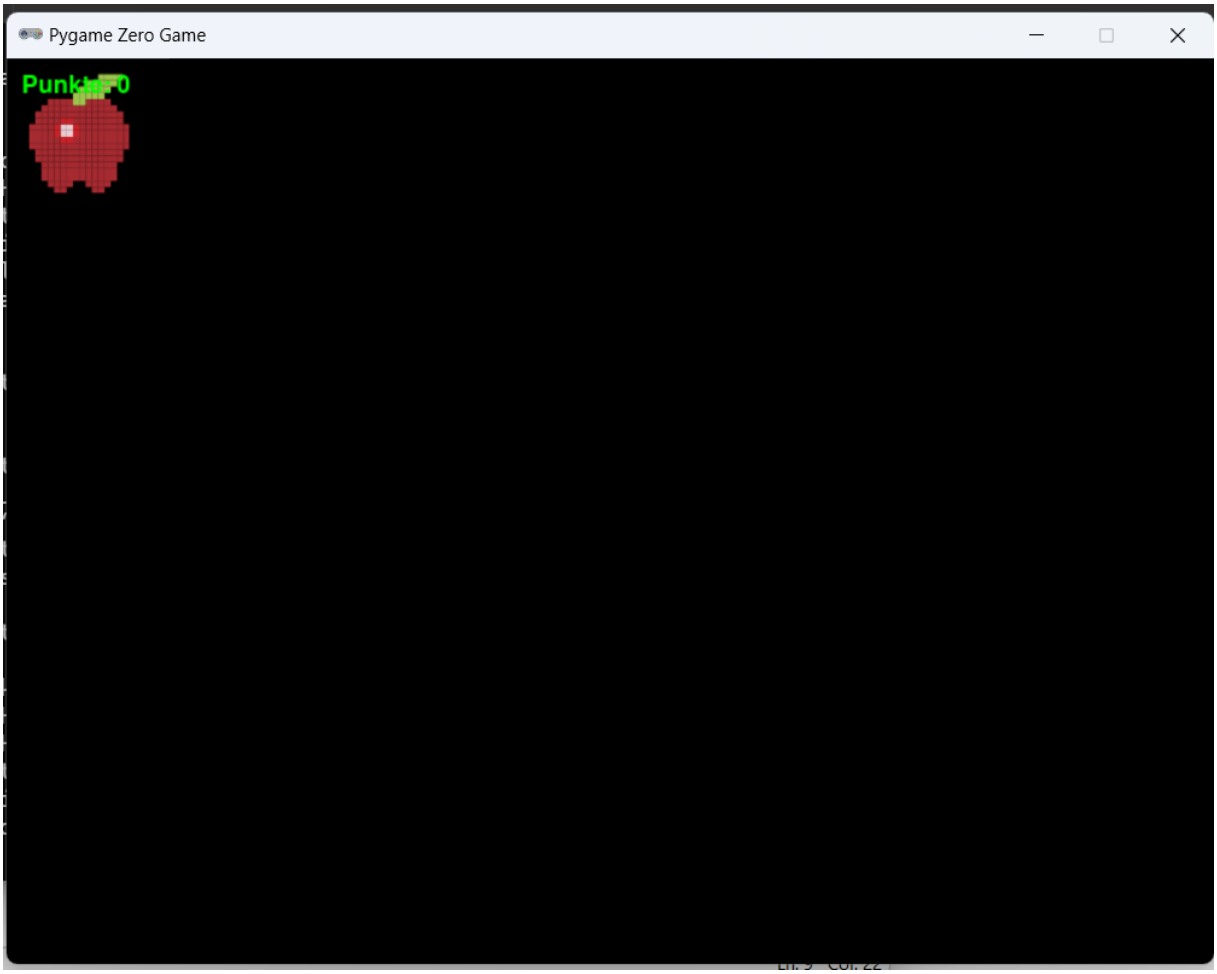
```
def draw():
    screen.clear()
    #Punktestand anzeigen
    screen.draw.text("Punkte: " + str(SCORE), color="green",
                     topleft=(10 ,10))
#Spiel beenden wenn der Timer agbelaufen ist und Punktestand
anzeigen
     if GAME_OVER:
        screen.fill("pink")
        screen.draw.text("Endstand: " + str(SCORE), topleft=(10,
                         10), fontsize=60)
clock.schedule(time_up, 300.0)
pgzrun.go()
```

Nach dem dies nun funktioniert, müssen wir noch einen Apfel auf unserem Spielfeld platzieren.

```
import pgzrun
WITH = 600
HEIGHT = 600
#Punktestand erstellen
SCORE = 0
GAME_OVER = False
#Apfel beschreiben
apple = Actor("apple")
def time_up():
    global GAME_OVER
    GAME_OVER = True
def draw():
    screen.clear()
    #Apfel zeichnen
    apple.draw()
    #Punktestand anzeigen
    screen.draw.text("Punkte: " + str(SCORE), color="green",
                     topleft=(10 ,10))
    if GAME_OVER:
        screen.fill("pink")
        screen.draw.text("Endstand: " + str(SCORE), topleft=(10,
```

```
        10), fontsize=60)
clock.schedule(time_up, 300.0)
pgzrun.go()
```

Sieht doch schon ganz gut aus 😊



Nun ist der Apfel aber immer an der gleichen Position, damit wir etwas mehr Spaß beim spielen haben, verschieben wir den Apfel nun um eine zufällige Anzahl auf dem Spielfeld.

Dazu müssen wir ein weiteres Modul importieren, das uns Zufallszahlen liefert.

```
import pgzrun
#Zufallszahlen importieren
from random import randint
WITH = 600
HEIGHT = 600
#Punktestand erstellen
SCORE = 0
GAME_OVER = False
#Apfel beschreiben
```

```
apple = Actor("apple")
def time_up():
    global GAME_OVER
    GAME_OVER = True
def draw():
    screen.clear()
    #Apfel zeichnen
    apple.draw()
    #Punktestand anzeigen
    screen.draw.text("Punkte: " + str(SCORE), color="green",
                     topleft=(10 ,10))
    if GAME_OVER:
        screen.fill("pink")
        screen.draw.text("Endstand: " + str(SCORE), topleft=(10,
                         10), fontsize=60)
#Funktion um den Apfel zufällig zu verschieben
def place_apple():
    apple.x = randint(10,WITH-20)
    apple.y = randint(10,HEIGHT-20)
clock.schedule(time_up, 300.0)
#Apfel verschieben
place_apple()
pgzrun.go()
```

Abschließend müssen wir den Apfel noch anklicken und auf das Klicken entsprechend reagieren, dazu benötigen wir eine weitere Funktion.

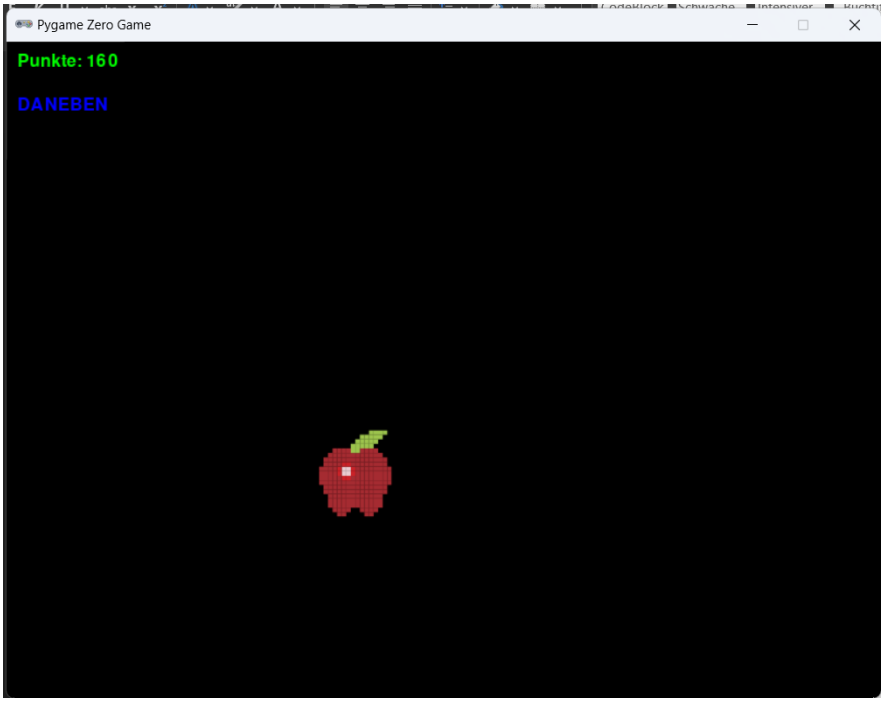
```
import pgzrun
#Zufallszahlen importieren
from random import randint
WITH = 600
HEIGHT = 600
#Punktestand erstellen
SCORE = 0
#Platzhalter für unsere Ausgabe
MESSAGE = ""
GAME_OVER = False
#Apfel beschreiben
```

```
apple = Actor("apple")

def time_up():
    global GAME_OVER
    GAME_OVER = True
def draw():
    screen.clear()
    #Apfel zeichnen
    apple.draw()
    #Punktestand anzeigen
    screen.draw.text("Punkte: " + str(SCORE), color="green",
                     topleft=(10 ,10))
    screen.draw.text(str(MESSAGE), color="blue", topleft=(10 ,50))
    if GAME_OVER:
        screen.fill("pink")
        screen.draw.text("Endstand: " + str(SCORE), topleft=(10,
                         10), fontsize=60)
#Apfel zufällig verschieben
def place_apple():
    apple.x = randint(10,WITH-20)
    apple.y = randint(10,HEIGHT-20)

#Funktion um auf die Mausklicks zu reagieren
def on_mouse_down(pos):
    global SCORE
    global GAME_OVER
    global MESSAGE
    if apple.collidepoint(pos):
        MESSAGE = "TREFFER - APFEL"
        SCORE += 10
        place_apple()
    else:
        MESSAGE = "DANEBEN"
clock.schedule(time_up, 300.0)
place_apple()
pgzrun.go()
```

FERTIG – Herzlichen Glückwunsch du hast dein erstes Spiel programmiert und nun schau mal wie du es noch erweitern kannst 😉



Verweise

Vorderman, C. a. (2019). *Spiele mit Python supereasy programmieren.* Dorling Kindersley Verlag GmbH.