Pygame Zero auf Pydroid - Teil 1: Deine ersten Spiele! 🎮

Hallo! In diesem Tutorial lernst du, wie du coole Spiele auf deinem Android-Handy programmieren kannst. Wir benutzen dafür **Pygame Zero** - das ist super einfach!

Was du am Ende kannst:

- Formen auf den Bildschirm malen
- Auf Touch-Eingaben reagieren
- 5 komplette Spiele programmieren!

Kapitel 1: Installation und Einrichtung

Ziel dieses Kapitels

Du installierst alles was du brauchst, um Spiele zu programmieren.

Was du brauchst

1. Ein Android-Handy oder Tablet
2. Die App **Pydroid 3** (aus dem Play Store)
3. Das Plugin **Pydroid repository plugin** (auch aus dem Play Store)

Schritt-für-Schritt Anleitung

Schritt 1: Pydroid 3 installieren

1. Öffne den Google Play Store auf deinem Handy
2. Suche nach "Pydroid 3"
3. Tippe auf "Installieren"
4. Warte bis die Installation fertig ist

Schritt 2: Das Plugin installieren

1. Suche im Play Store nach "Pydroid repository plugin"
2. Installiere es (das hilft dir später Pakete zu installieren)

Schritt 3: Pygame Zero installieren

1. Öffne Pydroid 3
2. Tippe auf die drei Striche oben links (≡)
3. Wähle "Pip" aus dem Menü
4. Gib in das Textfeld ein: `pgzero`
5. Tippe auf "Install"
6. Warte bis "Successfully installed" erscheint

Geschafft! Du bist bereit zum Programmieren!

Kapitel 2: Wie ein Pygame Zero Spiel funktioniert

Ziel dieses Kapitels

Du verstehst die Grundstruktur eines jeden Spiels.

Das musst du wissen

Jedes Spiel das du machst hat drei wichtige Teile:

```
┌─────────────────────────────────────────────┐
│  1. EINSTELLUNGEN (ganz oben)               │
│     → Wie groß ist das Spielfenster?        │
│     → Welche Variablen brauchen wir?        │
├─────────────────────────────────────────────┤
│  2. DRAW FUNKTION                           │
│     → Was soll auf dem Bildschirm sein?     │
│     → Wird ständig aufgerufen zum Zeichnen  │
├─────────────────────────────────────────────┤
│  3. UPDATE FUNKTION                         │
│     → Was soll sich verändern?              │
│     → Wird 60x pro Sekunde aufgerufen!      │
└─────────────────────────────────────────────┘
```

Das einfachste Spiel

```python
# TEIL 1: EINSTELLUNGEN
# Hier sagen wir wie groß das Fenster sein soll
WIDTH = 400   # Breite in Pixeln (links nach rechts)
HEIGHT = 600  # Höhe in Pixeln (oben nach unten)

# TEIL 2: DRAW FUNKTION
# Diese Funktion malt alles auf den Bildschirm
def draw():
    screen.fill("black")  # Hintergrund schwarz machen

# TEIL 3: UPDATE FUNKTION
# Diese Funktion wird immer wieder aufgerufen
def update():
    pass  # "pass" bedeutet: mach nichts
```

So startest du dein Spiel

Schritt 1: Öffne Pydroid 3

Schritt 2: Tippe auf das Ordner-Symbol und erstelle eine neue Datei

Schritt 3: Schreibe deinen Code

Schritt 4: Speichere die Datei mit der Endung `.py` (z.B. `meinspiel.py`)

Schritt 5: Tippe auf den Play-Knopf ▶️

Kapitel 3: Formen zeichnen lernen

Ziel dieses Kapitels

Du lernst verschiedene Formen auf den Bildschirm zu malen.

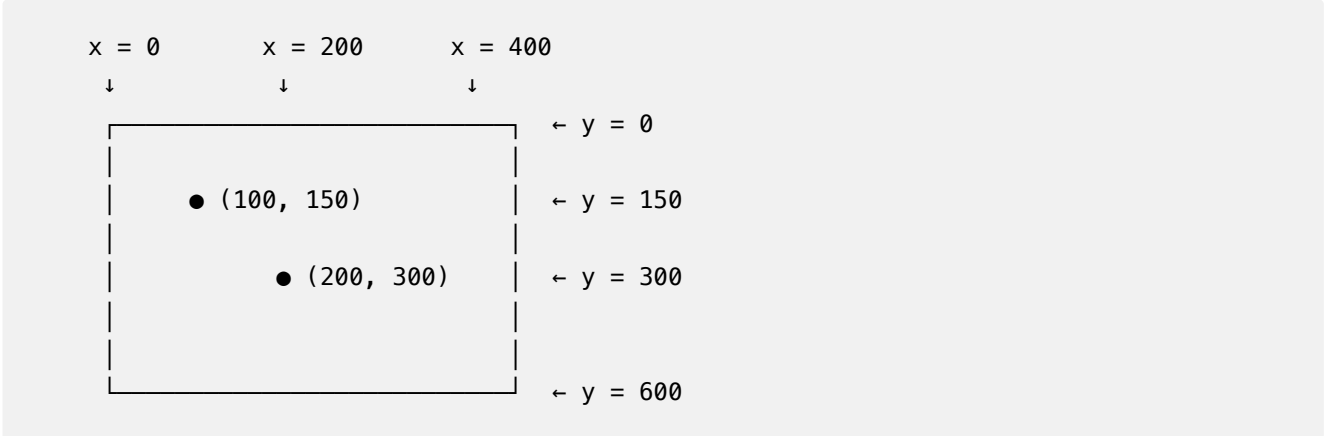
Warum ist das wichtig?

Bevor wir Spiele machen, müssen wir wissen wie man Dinge zeichnet. In unseren ersten Spielen benutzen wir einfache Formen wie Rechtecke und Kreise als Spielfiguren.

Das Koordinatensystem verstehen

Der Bildschirm ist wie ein Gitter aus Punkten. Jeder Punkt hat eine Position:

- **x** = wie weit nach rechts (0 ist ganz links)
- **y** = wie weit nach unten (0 ist ganz oben!)



Wichtig: y = 0 ist OBEN, nicht unten! Das ist am Anfang verwirrend, aber du gewöhnst dich daran.

Schritt 1: Ein Rechteck zeichnen

Ziel: Ein rotes Rechteck auf den Bildschirm malen.

So geht's:

```
# Rechteck zeichnen
# screen.draw.filled_rect(Rect(x, y, breite, höhe), farbe)
screen.draw.filled_rect(Rect(100, 200, 50, 50), "red")
```

Erklärung:

- `Rect(100, 200, 50, 50)` erstellt ein Rechteck
  - `100` = x-Position (100 Pixel von links)
  - `200` = y-Position (200 Pixel von oben)
  - `50` = Breite
  - `50` = Höhe
- `"red"` = die Farbe

Schritt 2: Einen Kreis zeichnen

Ziel: Einen gelben Kreis auf den Bildschirm malen.

So geht's:

```
# Kreis zeichnen
# screen.draw.filled_circle((x, y), radius, farbe)
screen.draw.filled_circle((200, 300), 30, "yellow")
```

Erklärung:

- `(200, 300)` = die Mitte des Kreises (x=200, y=300)

- `30` = der Radius (wie groß der Kreis ist)
- `"yellow"` = die Farbe

Schritt 3: Text schreiben

Ziel: Text auf den Bildschirm schreiben.

So geht's:

```
# Text schreiben
# screen.draw.text("dein text", (x, y), color="farbe", fontsize=größe)
screen.draw.text("Hallo!", (150, 100), color="white", fontsize=40)
```

Komplettes Beispiel: Alles zusammen

Ziel: Verschiedene Formen und Text auf einem blauen Hintergrund zeigen.

```
WIDTH = 400
HEIGHT = 600

def draw():
    # Schritt 1: Hintergrund blau machen
    screen.fill("darkblue")

    # Schritt 2: Ein rotes Rechteck zeichnen
    screen.draw.filled_rect(Rect(100, 200, 50, 50), "red")

    # Schritt 3: Ein grünes Rechteck zeichnen
    screen.draw.filled_rect(Rect(200, 200, 80, 40), "green")

    # Schritt 4: Einen gelben Kreis zeichnen
    screen.draw.filled_circle((200, 350), 40, "yellow")

    # Schritt 5: Einen orangen Kreis zeichnen
    screen.draw.filled_circle((100, 450), 25, "orange")

    # Schritt 6: Eine Linie zeichnen
    screen.draw.line((50, 500), (350, 500), "white")

    # Schritt 7: Text schreiben
    screen.draw.text("Hallo!", (150, 80), color="white", fontsize=50)
```

Übung für dich

Versuche folgendes zu ändern:

1. Mach den Hintergrund grün (`"green"`)
2. Mach das Rechteck größer (ändere 50, 50 zu 100, 100)
3. Füge deinen Namen als Text hinzu

Farben die du benutzen kannst

Englisch	Deutsch
`"red"`	Rot
`"blue"`	Blau
`"green"`	Grün
`"yellow"`	Gelb
`"white"`	Weiß
`"black"`	Schwarz
`"orange"`	Orange
`"purple"`	Lila
`"pink"`	Rosa
`"cyan"`	Türkis
`"gray"`	Grau
`"darkblue"`	Dunkelblau
`"darkgreen"`	Dunkelgrün
`"skyblue"`	Himmelblau

Kapitel 4: Touch-Eingabe verstehen

Ziel dieses Kapitels

Du lernst wie dein Spiel reagiert wenn jemand auf den Bildschirm tippt.

Warum ist das wichtig?

Bei Handyspielen steuerst du alles durch Tippen. Du musst wissen, wo der Finger den Bildschirm berührt hat.

Die spezielle Funktion: on_mouse_down

Wenn jemand auf den Bildschirm tippt, ruft Pygame Zero automatisch eine Funktion auf:

```
def on_mouse_down(pos):
    # pos enthält die Position wo getippt wurde
    # pos[0] = x-Position
    # pos[1] = y-Position
```

Das Problem mit Variablen

Wenn du eine Variable in einer Funktion ändern willst, musst du Python sagen, dass du die Variable von "außen" meinst. Das machst du mit dem Wort `global`.

Ohne global (funktioniert NICHT richtig):

```
punkte = 0

def on_mouse_down(pos):
    punkte = punkte + 1  # FEHLER! Python denkt das ist eine neue Variable
```

Mit global (funktioniert!):

```
punkte = 0

def on_mouse_down(pos):
    global punkte  # Sag Python: ich meine die Variable von oben!
    punkte = punkte + 1  # Jetzt geht es!
```

Schritt-für-Schritt: Ein Kreis folgt dem Finger

Ziel: Ein Kreis erscheint dort wo du auf den Bildschirm tippst.

Schritt 1: Die Grundstruktur erstellen

```
WIDTH = 400
HEIGHT = 600

# Startposition für den Kreis
kreis_x = 200
kreis_y = 300
```

Schritt 2: Den Kreis zeichnen

```
def draw():
    screen.fill("darkblue")
    screen.draw.filled_circle((kreis_x, kreis_y), 40, "red")
```

Schritt 3: Auf Tippen reagieren

```
def on_mouse_down(pos):
    global kreis_x, kreis_y  # WICHTIG: global nicht vergessen!
    kreis_x = pos[0]  # x-Position vom Finger
    kreis_y = pos[1]  # y-Position vom Finger
```

Das komplette Programm:

```
WIDTH = 400
HEIGHT = 600

# Hier speichern wir die Position des Kreises
kreis_x = 200
kreis_y = 300
anzahl_tipps = 0

def draw():
    screen.fill("darkblue")

    # Kreis an der aktuellen Position zeichnen
    screen.draw.filled_circle((kreis_x, kreis_y), 40, "red")

    # Anleitung zeigen
```

```
    screen.draw.text("Tippe irgendwo!", (100, 50), color="white", fontsize=30)

    # Position anzeigen
    screen.draw.text(f"X: {kreis_x}", (20, 520), color="yellow", fontsize=25)
    screen.draw.text(f"Y: {kreis_y}", (20, 550), color="yellow", fontsize=25)
    screen.draw.text(f"Tipps: {anzahl_tipps}", (250, 535), color="green", fontsize=25)

def on_mouse_down(pos):
    global kreis_x, kreis_y, anzahl_tipps
    kreis_x = pos[0]
    kreis_y = pos[1]
    anzahl_tipps = anzahl_tipps + 1
```

Was du gelernt hast

1. `on_mouse_down(pos)` wird aufgerufen wenn du tippst
2. `pos[0]` ist die x-Position (links/rechts)
3. `pos[1]` ist die y-Position (oben/unten)
4. `global` brauchst du um Variablen in Funktionen zu ändern

Kapitel 5: Bewegung mit der Update-Funktion

Ziel dieses Kapitels

Du lernst wie sich Dinge von alleine bewegen.

Warum ist das wichtig?

In Spielen bewegen sich Bälle, Feinde und andere Objekte automatisch. Das passiert alles in der `update()` Funktion.

Wie funktioniert update()?

Die `update()` Funktion wird 60 mal pro Sekunde aufgerufen! Wenn du dort die Position eines Objekts ein kleines bisschen änderst, sieht es aus als würde es sich bewegen.

```
Sekunde 1:    ●
Sekunde 2:      ●
Sekunde 3:        ●
Sekunde 4:          ●
              → Der Ball bewegt sich nach rechts!
```

Schritt-für-Schritt: Ein Ball der sich bewegt

Ziel: Ein Ball bewegt sich von links nach rechts über den Bildschirm.

Schritt 1: Variablen für Position und Geschwindigkeit

```
WIDTH = 400
HEIGHT = 600

ball_x = 50  # Startposition links
```

```
ball_y = 300  # Mitte des Bildschirms
geschwindigkeit = 3  # Wie viele Pixel pro Frame
```

Schritt 2: Den Ball zeichnen

```
def draw():
    screen.fill("darkblue")
    screen.draw.filled_circle((ball_x, ball_y), 20, "yellow")
```

Schritt 3: Den Ball bewegen

```
def update():
    global ball_x

    # Ball nach rechts bewegen
    ball_x = ball_x + geschwindigkeit

    # Wenn Ball rechts raus ist, links wieder rein
    if ball_x > WIDTH + 20:
        ball_x = -20
```

Das komplette Programm:

```
WIDTH = 400
HEIGHT = 600

ball_x = 50
ball_y = 300
geschwindigkeit = 3

def draw():
    screen.fill("darkblue")
    screen.draw.filled_circle((ball_x, ball_y), 20, "yellow")
    screen.draw.text("Der Ball bewegt sich!", (80, 50), color="white", fontsize=25)

def update():
    global ball_x
    ball_x = ball_x + geschwindigkeit

    if ball_x > WIDTH + 20:
        ball_x = -20
```

Schwerkraft simulieren

In vielen Spielen fallen Dinge nach unten. Das nennt man Schwerkraft.

Wie funktioniert Schwerkraft im Code?

```
geschwindigkeit_y = 0  # Am Anfang steht der Ball still
schwerkraft = 0.5      # Die Schwerkraft

def update():
```

```
    global ball_y, geschwindigkeit_y

    # Schwerkraft macht den Ball schneller (nach unten)
    geschwindigkeit_y = geschwindigkeit_y + schwerkraft

    # Ball fällt nach unten
    ball_y = ball_y + geschwindigkeit_y
```

SPIEL 1: Fang den Ball! ⚽

Ziel des Spiels

Ein Ball fällt von oben nach unten. Du musst ihn mit einem Fänger auffangen. Jedes Mal wenn du den Ball fängst, bekommst du einen Punkt!

Was wir programmieren müssen

```
┌─────────────────────────────────────────┐
│  1. Einen Ball der nach unten fällt     │
│  2. Einen Fänger den du steuern kannst  │
│  3. Punkte zählen                       │
│  4. Prüfen ob Ball gefangen wurde       │
│  5. Spiel schwerer machen               │
└─────────────────────────────────────────┘
```

Schritt 1: Die Grundstruktur

Ziel: Fenster erstellen und alle Variablen definieren.

```
import random  # Für Zufallszahlen

WIDTH = 400
HEIGHT = 600

# === DER BALL ===
ball_x = 200      # Startposition horizontal
ball_y = 0        # Startet oben
ball_speed = 5    # Wie schnell er fällt

# === DER FÄNGER ===
faenger_x = 175       # Startposition
faenger_y = 550       # Ganz unten
faenger_breite = 80   # Wie breit der Fänger ist
faenger_hoehe = 20    # Wie hoch der Fänger ist

# === SPIELSTAND ===
punkte = 0
```

Warum brauchen wir `import random`? Damit der Ball jedes Mal an einer anderen Stelle erscheint. `random.randint(1, 10)` gibt uns eine zufällige Zahl zwischen 1 und 10.

Schritt 2: Alles zeichnen

Ziel: Ball, Fänger und Punkte auf den Bildschirm malen.

```
def draw():
    # Hintergrund
    screen.fill("darkblue")

    # Ball zeichnen (ein gelber Kreis)
    screen.draw.filled_circle((ball_x, ball_y), 20, "yellow")

    # Fänger zeichnen (ein grünes Rechteck)
    screen.draw.filled_rect(
        Rect(faenger_x, faenger_y, faenger_breite, faenger_hoehe),
        "green"
    )

    # Punkte anzeigen
    screen.draw.text(f"Punkte: {punkte}", (10, 10), color="white", fontsize=30)
```

Schritt 3: Den Ball fallen lassen

Ziel: Der Ball soll von oben nach unten fallen.

```
def update():
    global ball_y

    # Ball fällt nach unten (y wird größer = weiter unten)
    ball_y = ball_y + ball_speed
```

Schritt 4: Prüfen ob Ball gefangen wurde

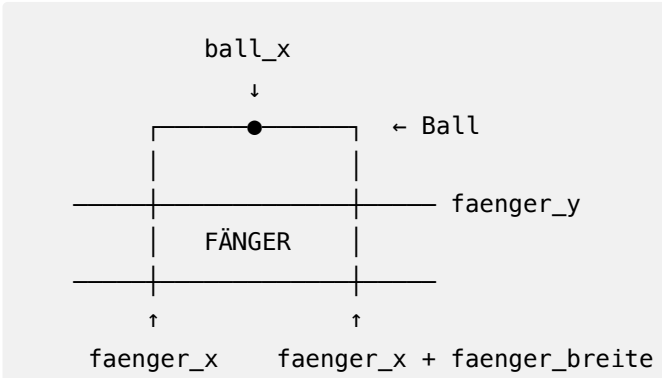
Ziel: Wenn der Ball den Fänger berührt, gibt es einen Punkt.

```
def update():
    global ball_x, ball_y, ball_speed, punkte

    # Ball fällt nach unten
    ball_y = ball_y + ball_speed

    # Ist der Ball auf Höhe des Fängers?
    if ball_y >= faenger_y:
        # Ist der Ball auch über dem Fänger (horizontal)?
        if ball_x >= faenger_x and ball_x <= faenger_x + faenger_breite:
            # GEFANGEN!
            punkte = punkte + 1
            # Ball zurück nach oben setzen
            ball_y = 0
            # Ball an zufälliger Position
            ball_x = random.randint(30, 370)
            # Spiel wird schneller!
            ball_speed = ball_speed + 0.5
```

Wie funktioniert die Kollisionsprüfung?



Wenn ball_x zwischen diesen beiden Werten ist = GEFANGEN!

Schritt 5: Ball nicht gefangen

Ziel: Wenn der Ball unten raus fällt, startet das Spiel neu.

```
    # Ball ist am Boden vorbei?
    if ball_y > HEIGHT:
        # Spiel neu starten
        ball_y = 0
        ball_x = random.randint(30, 370)
        punkte = 0  # Punkte zurücksetzen
        ball_speed = 5  # Geschwindigkeit zurücksetzen
```

Schritt 6: Fänger mit Finger steuern

Ziel: Wenn du auf den Bildschirm tippst, bewegt sich der Fänger dorthin.

```
def on_mouse_down(pos):
    global faenger_x
    # Fänger bewegt sich dahin wo du tippst
    # Wir ziehen die halbe Breite ab, damit die Mitte des Fängers beim Finger ist
    faenger_x = pos[0] - faenger_breite / 2
```

Das komplette Spiel

```
# === FANG DEN BALL ===
# Fange den Ball mit deinem Fänger!
# Tippe um den Fänger zu bewegen.

import random

WIDTH = 400
HEIGHT = 600

# Der Ball
ball_x = 200
ball_y = 0
ball_speed = 5
```

```
# Der Fänger (du steuerst ihn)
faenger_x = 175
faenger_y = 550
faenger_breite = 80
faenger_hoehe = 20

# Punkte
punkte = 0

def draw():
    screen.fill("darkblue")

    # Ball zeichnen
    screen.draw.filled_circle((ball_x, ball_y), 20, "yellow")

    # Fänger zeichnen
    screen.draw.filled_rect(
        Rect(faenger_x, faenger_y, faenger_breite, faenger_hoehe),
        "green"
    )

    # Punkte anzeigen
    screen.draw.text(f"Punkte: {punkte}", (10, 10), color="white", fontsize=30)
    screen.draw.text("Tippe um zu bewegen!", (80, 50), color="gray", fontsize=20)

def update():
    global ball_x, ball_y, ball_speed, punkte

    # Ball fällt nach unten
    ball_y = ball_y + ball_speed

    # Prüfen ob Ball gefangen wurde
    if ball_y >= faenger_y:
        if ball_x >= faenger_x and ball_x <= faenger_x + faenger_breite:
            # Getroffen! Punkt!
            punkte = punkte + 1
            ball_y = 0
            ball_x = random.randint(30, 370)
            ball_speed = ball_speed + 0.5

    # Ball ist unten raus?
    if ball_y > HEIGHT:
        ball_y = 0
        ball_x = random.randint(30, 370)
        punkte = 0
        ball_speed = 5

def on_mouse_down(pos):
    global faenger_x
    faenger_x = pos[0] - faenger_breite / 2
```

Ideen zum Verbessern

1. **Mach den Fänger schmaler** wenn du mehr Punkte hast
2. **Füge mehrere Bälle hinzu**
3. **Ändere die Farben** des Balls bei jedem Fang

SPIEL 2: Springende Box 📦

Ziel des Spiels

Eine Box läuft auf dem Boden. Hindernisse kommen von rechts. Du musst über sie springen! Wie das Dino-Spiel in Chrome.

Was wir programmieren müssen

```
┌─────────────────────────────────────────┐
│  1. Eine Box die auf dem Boden steht    │
│  2. Springen wenn du tippst             │
│  3. Schwerkraft (Box fällt zurück)      │
│  4. Hindernisse die sich bewegen        │
│  5. Kollision erkennen                  │
│  6. Game Over und Neustart              │
└─────────────────────────────────────────┘
```

Schritt 1: Die Grundstruktur

Ziel: Alle Variablen definieren.

```python
import random

WIDTH = 400
HEIGHT = 600

# === DIE BOX (der Spieler) ===
box_x = 80            # Position links
box_y = 450           # Startet auf dem Boden
box_breite = 40
box_hoehe = 40
box_speed_y = 0       # Geschwindigkeit nach oben/unten
schwerkraft = 0.8     # Zieht die Box nach unten
boden_y = 500         # Wo ist der Boden?

# === DAS HINDERNIS ===
hindernis_x = 400     # Startet rechts außerhalb
hindernis_breite = 40
hindernis_hoehe = 60

# === SPIELSTAND ===
punkte = 0
spiel_laeuft = True
geschwindigkeit = 6   # Wie schnell kommt das Hindernis?
```

Schritt 2: Alles zeichnen

Ziel: Box, Hindernis, Boden und Punkte malen.

```
def draw():
    # Himmel
    screen.fill("skyblue")

    # Boden (grünes Rechteck ganz unten)
    screen.draw.filled_rect(Rect(0, boden_y, WIDTH, 100), "green")

    # Box (der Spieler)
    screen.draw.filled_rect(
        Rect(box_x, box_y, box_breite, box_hoehe),
        "red"
    )

    # Hindernis
    screen.draw.filled_rect(
        Rect(hindernis_x, boden_y - hindernis_hoehe, hindernis_breite, hindernis_hoehe),
        "darkgreen"
    )

    # Punkte
    screen.draw.text(f"Punkte: {punkte}", (10, 10), color="black", fontsize=30)

    # Anleitung
    screen.draw.text("Tippe zum Springen!", (100, 50), color="gray", fontsize=20)

    # Game Over Nachricht
    if not spiel_laeuft:
        screen.draw.text("GAME OVER!", (100, 250), color="red", fontsize=50)
        screen.draw.text("Tippe zum Neustarten", (80, 320), color="black", fontsize=25)
```

Schritt 3: Schwerkraft und Springen

Ziel: Die Box soll fallen und springen können.

Wie funktioniert Springen?

```
1. Du tippst → box_speed_y wird negativ (z.B. -15)
2. Box fliegt nach oben (y wird kleiner)
3. Schwerkraft macht box_speed_y langsam positiv
4. Box fängt an zu fallen
5. Box landet auf dem Boden
```

```
def update():
    global box_y, box_speed_y

    # Schwerkraft - macht die Box langsamer (beim Hochfliegen)
    # und schneller (beim Fallen)
    box_speed_y = box_speed_y + schwerkraft

    # Box bewegen
```

```
    box_y = box_y + box_speed_y

    # Box darf nicht durch den Boden fallen!
    if box_y + box_hoehe > boden_y:
        box_y = boden_y - box_hoehe  # Auf den Boden setzen
        box_speed_y = 0              # Stoppen
```

Schritt 4: Hindernis bewegen

Ziel: Das Hindernis bewegt sich von rechts nach links.

```
def update():
    global hindernis_x, punkte, geschwindigkeit
    # (... vorheriger Code ...)

    # Hindernis bewegt sich nach links
    hindernis_x = hindernis_x - geschwindigkeit

    # Hindernis ist links raus? Neues Hindernis!
    if hindernis_x < -hindernis_breite:
        hindernis_x = WIDTH + random.randint(0, 200)  # Rechts neu starten
        punkte = punkte + 1  # Punkt!

        # Spiel wird schneller
        if geschwindigkeit < 15:
            geschwindigkeit = geschwindigkeit + 0.3
```

Schritt 5: Kollision prüfen

Ziel: Erkennen wenn die Box das Hindernis berührt.

Wie funktioniert Kollision? Zwei Rechtecke überlappen sich wenn:

- Sie sich horizontal überlappen UND
- Sie sich vertikal überlappen

```
def update():
    global spiel_laeuft
    # (... vorheriger Code ...)

    # Kollision prüfen
    hindernis_y = boden_y - hindernis_hoehe

    # Überlappen sich Box und Hindernis horizontal?
    if box_x + box_breite > hindernis_x and box_x < hindernis_x + hindernis_breite:
        # Überlappen sie sich auch vertikal?
        if box_y + box_hoehe > hindernis_y:
            # GETROFFEN!
            spiel_laeuft = False
```

Schritt 6: Springen und Neustart

Ziel: Tippen = Springen (oder Neustart nach Game Over)

```
def on_mouse_down(pos):
    global box_speed_y, spiel_laeuft, punkte, hindernis_x, geschwindigkeit

    if spiel_laeuft:
        # Nur springen wenn Box auf dem Boden ist
        if box_y + box_hoehe >= boden_y - 5:
            box_speed_y = -15  # Nach oben springen!
    else:
        # Spiel neu starten
        spiel_laeuft = True
        punkte = 0
        hindernis_x = 400
        geschwindigkeit = 6
```

Das komplette Spiel

```
# === SPRINGENDE BOX ===
# Springe über die Hindernisse!
# Tippe um zu springen.

import random

WIDTH = 400
HEIGHT = 600

# Die Box (der Spieler)
box_x = 80
box_y = 450
box_breite = 40
box_hoehe = 40
box_speed_y = 0
schwerkraft = 0.8
boden_y = 500

# Hindernis
hindernis_x = 400
hindernis_breite = 40
hindernis_hoehe = 60

# Spielstand
punkte = 0
spiel_laeuft = True
geschwindigkeit = 6

def draw():
    screen.fill("skyblue")

    # Boden
    screen.draw.filled_rect(Rect(0, boden_y, WIDTH, 100), "green")

    # Box
    screen.draw.filled_rect(
```

```
        Rect(box_x, box_y, box_breite, box_hoehe),
        "red"
    )

    # Hindernis
    screen.draw.filled_rect(
        Rect(hindernis_x, boden_y - hindernis_hoehe, hindernis_breite, hindernis_hoehe),
        "darkgreen"
    )

    # Punkte
    screen.draw.text(f"Punkte: {punkte}", (10, 10), color="black", fontsize=30)
    screen.draw.text("Tippe zum Springen!", (100, 50), color="gray", fontsize=20)

    # Game Over
    if not spiel_laeuft:
        screen.draw.text("GAME OVER!", (100, 250), color="red", fontsize=50)
        screen.draw.text("Tippe zum Neustarten", (80, 320), color="black", fontsize=25)

def update():
    global box_y, box_speed_y, hindernis_x, punkte, spiel_laeuft, geschwindigkeit

    if not spiel_laeuft:
        return

    # Schwerkraft
    box_speed_y = box_speed_y + schwerkraft
    box_y = box_y + box_speed_y

    # Boden-Kollision
    if box_y + box_hoehe > boden_y:
        box_y = boden_y - box_hoehe
        box_speed_y = 0

    # Hindernis bewegen
    hindernis_x = hindernis_x - geschwindigkeit

    # Neues Hindernis
    if hindernis_x < -hindernis_breite:
        hindernis_x = WIDTH + random.randint(0, 200)
        punkte = punkte + 1
        if geschwindigkeit < 15:
            geschwindigkeit = geschwindigkeit + 0.3

    # Kollision mit Hindernis
    hindernis_y = boden_y - hindernis_hoehe
    if box_x + box_breite > hindernis_x and box_x < hindernis_x + hindernis_breite:
        if box_y + box_hoehe > hindernis_y:
            spiel_laeuft = False

def on_mouse_down(pos):
    global box_speed_y, spiel_laeuft, punkte, hindernis_x, geschwindigkeit

    if spiel_laeuft:
```

```
        if box_y + box_hoehe >= boden_y - 5:
            box_speed_y = -15
    else:
        spiel_laeuft = True
        punkte = 0
        hindernis_x = 400
        geschwindigkeit = 6
```

SPIEL 3: Flappy Box 🐦

Ziel des Spiels

Eine Box fliegt durch Röhren. Tippe um nach oben zu fliegen. Berühre keine Röhre!

Was wir programmieren müssen

```
┌─────────────────────────────────────────┐
│  1. Eine Box die fällt                  │
│  2. Tippen = nach oben fliegen          │
│  3. Röhren mit Lücke                    │
│  4. Röhren bewegen sich nach links      │
│  5. Kollision mit Röhren                │
│  6. Punkte und Game Over                │
└─────────────────────────────────────────┘
```

Schritt 1: Grundstruktur und Box

```
import random

WIDTH = 400
HEIGHT = 600

# Die Box
box_x = 100           # Bleibt immer gleich (links)
box_y = 300           # Startet in der Mitte
box_groesse = 30
box_speed_y = 0       # Aktuelle Geschwindigkeit
schwerkraft = 0.5     # Zieht nach unten
sprung_kraft = -10    # Wie stark nach oben beim Tippen
```

Schritt 2: Die Röhren

Ziel: Zwei Röhren (oben und unten) mit einer Lücke dazwischen.

```
┌──────────────┐
│   OBEN       │  ← obere Röhre (von y=0 bis luecke_y)
│              │
└──────────────┘

                 ← LÜCKE (luecke_hoehe groß)

┌──────────────┐
```

```
|   UNTEN         |  ← untere Röhre (von luecke_y+luecke_hoehe bis unten)
|                 |
└─────────────────┘
```

```
# Röhren
roehre_x = 400        # Startet rechts
luecke_y = 250        # Wo die Lücke anfängt
luecke_hoehe = 180    # Wie groß die Lücke ist (größer = einfacher!)
roehre_breite = 60
roehre_speed = 4

# Spielstand
punkte = 0
spiel_laeuft = True
```

Schritt 3: Zeichnen

```
def draw():
    screen.fill("skyblue")

    # Obere Röhre (von ganz oben bis zur Lücke)
    screen.draw.filled_rect(
        Rect(roehre_x, 0, roehre_breite, luecke_y),
        "green"
    )

    # Untere Röhre (von Ende der Lücke bis ganz unten)
    untere_roehre_y = luecke_y + luecke_hoehe
    screen.draw.filled_rect(
        Rect(roehre_x, untere_roehre_y, roehre_breite, HEIGHT - untere_roehre_y),
        "green"
    )

    # Box
    screen.draw.filled_rect(
        Rect(box_x, box_y, box_groesse, box_groesse),
        "yellow"
    )

    # Punkte (groß in der Mitte)
    screen.draw.text(f"{punkte}", (WIDTH/2 - 20, 50), color="white", fontsize=60)

    # Game Over
    if not spiel_laeuft:
        screen.draw.text("GAME OVER", (80, 250), color="red", fontsize=50)
        screen.draw.text("Tippe zum Neustarten", (70, 320), color="black", fontsize=25)
```

Schritt 4: Bewegung und Kollision

```
def update():
    global box_y, box_speed_y, roehre_x, luecke_y, punkte, spiel_laeuft
```

```
    if not spiel_laeuft:
        return

    # Schwerkraft – Box fällt nach unten
    box_speed_y = box_speed_y + schwerkraft
    box_y = box_y + box_speed_y

    # Röhre bewegt sich nach links
    roehre_x = roehre_x - roehre_speed

    # Neue Röhre wenn alte weg ist
    if roehre_x < -roehre_breite:
        roehre_x = WIDTH
        # Lücke an zufälliger Höhe
        luecke_y = random.randint(80, HEIGHT - luecke_hoehe - 80)
        punkte = punkte + 1

    # KOLLISION MIT RÖHREN
    # Ist die Box auf gleicher x-Position wie die Röhre?
    if box_x + box_groesse > roehre_x and box_x < roehre_x + roehre_breite:
        # Ist die Box NICHT in der Lücke?
        if box_y < luecke_y or box_y + box_groesse > luecke_y + luecke_hoehe:
            spiel_laeuft = False

    # Kollision mit Boden oder Decke
    if box_y < 0 or box_y + box_groesse > HEIGHT:
        spiel_laeuft = False
```

Schritt 5: Steuerung

```
def on_mouse_down(pos):
    global box_speed_y, spiel_laeuft, box_y, roehre_x, punkte

    if spiel_laeuft:
        # Nach oben fliegen!
        box_speed_y = sprung_kraft
    else:
        # Neustart
        spiel_laeuft = True
        box_y = 300
        box_speed_y = 0
        roehre_x = 400
        punkte = 0
```

Das komplette Spiel

```
# === FLAPPY BOX ===
# Fliege durch die Lücken!
# Tippe um nach oben zu fliegen.

import random
```

```
WIDTH = 400
HEIGHT = 600

# Die Box
box_x = 100
box_y = 300
box_groesse = 30
box_speed_y = 0
schwerkraft = 0.5
sprung_kraft = -10

# Röhren
roehre_x = 400
luecke_y = 250
luecke_hoehe = 180
roehre_breite = 60
roehre_speed = 4

# Spielstand
punkte = 0
spiel_laeuft = True

def draw():
    screen.fill("skyblue")

    # Obere Röhre
    screen.draw.filled_rect(
        Rect(roehre_x, 0, roehre_breite, luecke_y),
        "green"
    )

    # Untere Röhre
    untere_roehre_y = luecke_y + luecke_hoehe
    screen.draw.filled_rect(
        Rect(roehre_x, untere_roehre_y, roehre_breite, HEIGHT - untere_roehre_y),
        "green"
    )

    # Box
    screen.draw.filled_rect(
        Rect(box_x, box_y, box_groesse, box_groesse),
        "yellow"
    )

    # Punkte
    screen.draw.text(f"{punkte}", (WIDTH/2 - 20, 50), color="white", fontsize=60)

    if not spiel_laeuft:
        screen.draw.text("GAME OVER", (80, 250), color="red", fontsize=50)
        screen.draw.text("Tippe zum Neustarten", (70, 320), color="black", fontsize=25)

def update():
    global box_y, box_speed_y, roehre_x, luecke_y, punkte, spiel_laeuft
```

```
    if not spiel_laeuft:
        return

    box_speed_y = box_speed_y + schwerkraft
    box_y = box_y + box_speed_y

    roehre_x = roehre_x - roehre_speed

    if roehre_x < -roehre_breite:
        roehre_x = WIDTH
        luecke_y = random.randint(80, HEIGHT - luecke_hoehe - 80)
        punkte = punkte + 1

    if box_x + box_groesse > roehre_x and box_x < roehre_x + roehre_breite:
        if box_y < luecke_y or box_y + box_groesse > luecke_y + luecke_hoehe:
            spiel_laeuft = False

    if box_y < 0 or box_y + box_groesse > HEIGHT:
        spiel_laeuft = False

def on_mouse_down(pos):
    global box_speed_y, spiel_laeuft, box_y, roehre_x, punkte

    if spiel_laeuft:
        box_speed_y = sprung_kraft
    else:
        spiel_laeuft = True
        box_y = 300
        box_speed_y = 0
        roehre_x = 400
        punkte = 0
```

SPIEL 4: Ping Pong 🏓

Ziel des Spiels

Ein Ball springt hin und her. Du spielst gegen den Computer. Triff den Ball mit deinem Schläger!

Was wir programmieren müssen

```
┌──────────────────────────────────────┐
│  1. Einen Ball der sich bewegt       │
│  2. Ball prallt von Wänden ab        │
│  3. Dein Schläger (unten)            │
│  4. Computer Schläger (oben)         │
│  5. Ball trifft Schläger = abprallen │
│  6. Ball verfehlt = Punkt für Gegner │
└──────────────────────────────────────┘
```

Schritt 1: Ball-Bewegung

Der Ball hat zwei Geschwindigkeiten:

- `ball_speed_x` = Bewegung nach links/rechts
- `ball_speed_y` = Bewegung nach oben/unten

```
WIDTH = 400
HEIGHT = 600

# Der Ball
ball_x = 200
ball_y = 300
ball_groesse = 15
ball_speed_x = 5   # Positiv = nach rechts, negativ = nach links
ball_speed_y = 5   # Positiv = nach unten, negativ = nach oben
```

Schritt 2: Schläger

```
# Spieler Schläger (unten)
spieler_x = 150
spieler_y = 550
schlaeger_breite = 100
schlaeger_hoehe = 15

# Computer Schläger (oben)
computer_x = 150
computer_y = 30
computer_speed = 4  # Wie schnell der Computer reagiert

# Punkte
spieler_punkte = 0
computer_punkte = 0
```

Schritt 3: Ball bewegen und abprallen

```
def update():
    global ball_x, ball_y, ball_speed_x, ball_speed_y
    global spieler_punkte, computer_punkte

    # Ball bewegen
    ball_x = ball_x + ball_speed_x
    ball_y = ball_y + ball_speed_y

    # Ball prallt von linker und rechter Wand ab
    if ball_x <= 0 or ball_x + ball_groesse >= WIDTH:
        ball_speed_x = -ball_speed_x  # Richtung umkehren!
```

Was bedeutet `-ball_speed_x`? Wenn `ball_speed_x = 5` (nach rechts), dann wird es `-5` (nach links). Der Ball fliegt in die andere Richtung!

Schritt 4: Ball trifft Schläger

```
    # Ball trifft SPIELER Schläger (unten)
    if ball_y + ball_groesse >= spieler_y:
        # Ist der Ball auch über dem Schläger?
        if ball_x + ball_groesse >= spieler_x and ball_x <= spieler_x + schlaeger_breite:
            ball_speed_y = -abs(ball_speed_y)  # Nach oben fliegen

    # Ball trifft COMPUTER Schläger (oben)
    if ball_y <= computer_y + schlaeger_hoehe:
        if ball_x + ball_groesse >= computer_x and ball_x <= computer_x + schlaeger_breit
            ball_speed_y = abs(ball_speed_y)  # Nach unten fliegen
```

Was bedeutet `abs()`? `abs()` macht eine Zahl immer positiv.

- `abs(-5)` = 5
- `abs(5)` = 5

So stellen wir sicher, dass der Ball in die richtige Richtung fliegt.

Schritt 5: Punkte vergeben

```
    # Ball geht OBEN raus = Punkt für Spieler
    if ball_y < 0:
        spieler_punkte = spieler_punkte + 1
        ball_x = 200  # Ball zurück zur Mitte
        ball_y = 300

    # Ball geht UNTEN raus = Punkt für Computer
    if ball_y > HEIGHT:
        computer_punkte = computer_punkte + 1
        ball_x = 200
        ball_y = 300
```

Schritt 6: Computer Schläger (KI)

Der Computer folgt einfach dem Ball:

```
    # Computer KI - folgt dem Ball
    ball_mitte = ball_x + ball_groesse / 2
    computer_mitte = computer_x + schlaeger_breite / 2

    # Ball ist links vom Computer? → nach links bewegen
    if ball_mitte < computer_mitte - 10:
        computer_x = computer_x - computer_speed
    # Ball ist rechts vom Computer? → nach rechts bewegen
    elif ball_mitte > computer_mitte + 10:
        computer_x = computer_x + computer_speed
```

Schritt 7: Spieler steuern

```
def on_mouse_down(pos):
    global spieler_x
    spieler_x = pos[0] - schlaeger_breite / 2
```

```
    # Nicht aus dem Bildschirm!
    if spieler_x < 0:
        spieler_x = 0
    if spieler_x > WIDTH - schlaeger_breite:
        spieler_x = WIDTH - schlaeger_breite

def on_mouse_move(pos):
    global spieler_x
    spieler_x = pos[0] - schlaeger_breite / 2

    if spieler_x < 0:
        spieler_x = 0
    if spieler_x > WIDTH - schlaeger_breite:
        spieler_x = WIDTH - schlaeger_breite
```

Was ist `on_mouse_move`? Diese Funktion wird aufgerufen wenn du deinen Finger über den Bildschirm ziehst (ohne loszulassen).

Das komplette Spiel

```
# === PING PONG ===
# Spiele gegen den Computer!
# Tippe oder ziehe um deinen Schläger zu bewegen.

WIDTH = 400
HEIGHT = 600

# Der Ball
ball_x = 200
ball_y = 300
ball_groesse = 15
ball_speed_x = 5
ball_speed_y = 5

# Spieler Schläger (unten - grün)
spieler_x = 150
spieler_y = 550
schlaeger_breite = 100
schlaeger_hoehe = 15

# Computer Schläger (oben - rot)
computer_x = 150
computer_y = 30
computer_speed = 4

# Punkte
spieler_punkte = 0
computer_punkte = 0

def draw():
    screen.fill("black")
```

```
    # Mittellinie
    screen.draw.line((0, HEIGHT/2), (WIDTH, HEIGHT/2), "gray")

    # Ball
    screen.draw.filled_rect(
        Rect(ball_x, ball_y, ball_groesse, ball_groesse),
        "white"
    )

    # Spieler Schläger
    screen.draw.filled_rect(
        Rect(spieler_x, spieler_y, schlaeger_breite, schlaeger_hoehe),
        "green"
    )

    # Computer Schläger
    screen.draw.filled_rect(
        Rect(computer_x, computer_y, schlaeger_breite, schlaeger_hoehe),
        "red"
    )

    # Punkte
    screen.draw.text(f"Computer: {computer_punkte}", (10, HEIGHT/2 - 40),
                     color="red", fontsize=25)
    screen.draw.text(f"Du: {spieler_punkte}", (10, HEIGHT/2 + 10),
                     color="green", fontsize=25)

def update():
    global ball_x, ball_y, ball_speed_x, ball_speed_y
    global computer_x, spieler_punkte, computer_punkte

    # Ball bewegen
    ball_x = ball_x + ball_speed_x
    ball_y = ball_y + ball_speed_y

    # Ball prallt von Seiten ab
    if ball_x <= 0 or ball_x + ball_groesse >= WIDTH:
        ball_speed_x = -ball_speed_x

    # Ball trifft Spieler Schläger
    if ball_y + ball_groesse >= spieler_y:
        if ball_x + ball_groesse >= spieler_x and ball_x <= spieler_x + schlaeger_breite:
            ball_speed_y = -abs(ball_speed_y)

    # Ball trifft Computer Schläger
    if ball_y <= computer_y + schlaeger_hoehe:
        if ball_x + ball_groesse >= computer_x and ball_x <= computer_x + schlaeger_breit
            ball_speed_y = abs(ball_speed_y)

    # Punkt für Spieler
    if ball_y < 0:
        spieler_punkte = spieler_punkte + 1
        ball_x = 200
        ball_y = 300
```

```
    # Punkt für Computer
    if ball_y > HEIGHT:
        computer_punkte = computer_punkte + 1
        ball_x = 200
        ball_y = 300

    # Computer KI
    ball_mitte = ball_x + ball_groesse / 2
    computer_mitte = computer_x + schlaeger_breite / 2

    if ball_mitte < computer_mitte - 10:
        computer_x = computer_x - computer_speed
    elif ball_mitte > computer_mitte + 10:
        computer_x = computer_x + computer_speed

def on_mouse_down(pos):
    global spieler_x
    spieler_x = pos[0] - schlaeger_breite / 2

    if spieler_x < 0:
        spieler_x = 0
    if spieler_x > WIDTH - schlaeger_breite:
        spieler_x = WIDTH - schlaeger_breite

def on_mouse_move(pos):
    global spieler_x
    spieler_x = pos[0] - schlaeger_breite / 2

    if spieler_x < 0:
        spieler_x = 0
    if spieler_x > WIDTH - schlaeger_breite:
        spieler_x = WIDTH - schlaeger_breite
```

SPIEL 5: Breakout 

Ziel des Spiels

Zerstöre alle Steine mit dem Ball! Lass den Ball nicht fallen.

Was wir programmieren müssen

```
┌──────────────────────────────────────┐
│  1. Viele Steine oben                │
│  2. Ein Ball der sich bewegt         │
│  3. Ein Schläger unten               │
│  4. Ball zerstört Steine             │
│  5. Leben verlieren wenn Ball fällt  │
│  6. Gewinnen wenn alle Steine weg sind  │
└──────────────────────────────────────┘
```

Schritt 1: Steine mit einer Liste

Ziel: Viele Steine erstellen und in einer Liste speichern.

Was ist eine Liste? Eine Liste speichert viele Dinge zusammen:

```
meine_zahlen = [1, 2, 3, 4, 5]
meine_farben = ["rot", "blau", "grün"]
```

Für die Steine benutzen wir eine Liste mit Dictionaries:

```
steine = [
    {"x": 10, "y": 60, "farbe": "red"},
    {"x": 60, "y": 60, "farbe": "red"},
    # ... und so weiter
]
```

Schritt 2: Steine automatisch erstellen

```
def erstelle_steine():
    global steine
    steine = []  # Leere Liste
    farben = ["red", "orange", "yellow", "green", "blue"]

    # 5 Reihen
    for reihe in range(5):
        # 8 Steine pro Reihe
        for spalte in range(8):
            x = spalte * (stein_breite + stein_abstand) + 10
            y = reihe * (stein_hoehe + stein_abstand) + 60
            farbe = farben[reihe]  # Jede Reihe andere Farbe
            steine.append({"x": x, "y": y, "farbe": farbe})

erstelle_steine()  # Am Anfang aufrufen!
```

Was macht `for reihe in range(5)`? Das macht eine Schleife die 5 mal läuft:

- Erstes mal: reihe = 0
- Zweites mal: reihe = 1
- ...
- Fünftes mal: reihe = 4

Schritt 3: Ball trifft Stein

```
def update():
    global punkte, ball_speed_y
    # (... Ball-Bewegung Code ...)

    # Ball trifft Steine
    for stein in steine[:]:  # [:] macht eine Kopie der Liste
        # Überlappen sich Ball und Stein?
        if (ball_x + ball_groesse > stein["x"] and
```

```
            ball_x - ball_groesse < stein["x"] + stein_breite and
            ball_y + ball_groesse > stein["y"] and
            ball_y - ball_groesse < stein["y"] + stein_hoehe):
            # Stein getroffen!
            steine.remove(stein)  # Stein entfernen
            ball_speed_y = -ball_speed_y  # Ball prallt ab
            punkte = punkte + 10
            break  # Nur ein Stein pro Frame
```

Warum `steine[:]`? Wenn du Elemente aus einer Liste entfernst während du durch sie gehst, kann es Probleme geben. `steine[:]` macht eine Kopie der Liste, damit das nicht passiert.

Das komplette Spiel

```
# === BREAKOUT ===
# Zerstöre alle Steine!
# Tippe oder ziehe um den Schläger zu bewegen.

WIDTH = 400
HEIGHT = 600

# Ball
ball_x = 200
ball_y = 400
ball_groesse = 12
ball_speed_x = 4
ball_speed_y = -4

# Schläger
schlaeger_x = 150
schlaeger_y = 550
schlaeger_breite = 80
schlaeger_hoehe = 12

# Steine
steine = []
stein_breite = 45
stein_hoehe = 20
stein_abstand = 5

# Spielstand
punkte = 0
leben = 3

def erstelle_steine():
    global steine
    steine = []
    farben = ["red", "orange", "yellow", "green", "blue"]

    for reihe in range(5):
        for spalte in range(8):
            x = spalte * (stein_breite + stein_abstand) + 10
            y = reihe * (stein_hoehe + stein_abstand) + 60
```

```
            farbe = farben[reihe]
            steine.append({"x": x, "y": y, "farbe": farbe})

erstelle_steine()

def draw():
    screen.fill("black")

    # Alle Steine zeichnen
    for stein in steine:
        screen.draw.filled_rect(
            Rect(stein["x"], stein["y"], stein_breite, stein_hoehe),
            stein["farbe"]
        )

    # Ball
    screen.draw.filled_circle((ball_x, ball_y), ball_groesse, "white")

    # Schläger
    screen.draw.filled_rect(
        Rect(schlaeger_x, schlaeger_y, schlaeger_breite, schlaeger_hoehe),
        "cyan"
    )

    # Info
    screen.draw.text(f"Punkte: {punkte}", (10, 10), color="white", fontsize=25)
    screen.draw.text(f"Leben: {leben}", (300, 10), color="white", fontsize=25)

    # Gewonnen?
    if len(steine) == 0:
        screen.draw.text("DU HAST GEWONNEN!", (50, 300), color="green", fontsize=35)

    # Verloren?
    if leben <= 0:
        screen.draw.text("GAME OVER", (100, 300), color="red", fontsize=45)

def update():
    global ball_x, ball_y, ball_speed_x, ball_speed_y, punkte, leben

    if leben <= 0 or len(steine) == 0:
        return

    # Ball bewegen
    ball_x = ball_x + ball_speed_x
    ball_y = ball_y + ball_speed_y

    # Ball prallt von Wänden ab
    if ball_x - ball_groesse <= 0 or ball_x + ball_groesse >= WIDTH:
        ball_speed_x = -ball_speed_x

    # Ball prallt von Decke ab
    if ball_y - ball_groesse <= 0:
        ball_speed_y = -ball_speed_y
```

```
    # Ball fällt runter
    if ball_y > HEIGHT:
        leben = leben - 1
        ball_x = 200
        ball_y = 400
        ball_speed_y = -4

    # Ball trifft Schläger
    if ball_y + ball_groesse >= schlaeger_y:
        if ball_x >= schlaeger_x and ball_x <= schlaeger_x + schlaeger_breite:
            ball_speed_y = -abs(ball_speed_y)

    # Ball trifft Steine
    for stein in steine[:]:
        if (ball_x + ball_groesse > stein["x"] and
            ball_x - ball_groesse < stein["x"] + stein_breite and
            ball_y + ball_groesse > stein["y"] and
            ball_y - ball_groesse < stein["y"] + stein_hoehe):
            steine.remove(stein)
            ball_speed_y = -ball_speed_y
            punkte = punkte + 10
            break

def on_mouse_down(pos):
    global schlaeger_x
    schlaeger_x = pos[0] - schlaeger_breite / 2

    if schlaeger_x < 0:
        schlaeger_x = 0
    if schlaeger_x > WIDTH - schlaeger_breite:
        schlaeger_x = WIDTH - schlaeger_breite

def on_mouse_move(pos):
    global schlaeger_x
    schlaeger_x = pos[0] - schlaeger_breite / 2

    if schlaeger_x < 0:
        schlaeger_x = 0
    if schlaeger_x > WIDTH - schlaeger_breite:
        schlaeger_x = WIDTH - schlaeger_breite
```

Bonus: Bildschirm-Buttons erstellen 🔘

Ziel dieses Kapitels

Du lernst Buttons auf den Bildschirm zu zeichnen die du antippen kannst.

Warum ist das nützlich?

Manchmal willst du Steuerungstasten auf dem Bildschirm haben, zum Beispiel Pfeiltasten für links/rechts/hoch/runter.

Wie funktioniert es?

1. **Button als Rechteck definieren** (mit Position und Größe)
2. **Button zeichnen** (mit `filled_rect`)
3. **Prüfen ob getippt wurde** (mit `collidepoint`)

Schritt-für-Schritt Beispiel

```
# === SPIEL MIT BUTTONS ===

WIDTH = 400
HEIGHT = 600

# Spieler Position
spieler_x = 180
spieler_y = 250

# Buttons definieren - Rect(x, y, breite, höhe)
button_links = Rect(20, 480, 80, 60)
button_rechts = Rect(120, 480, 80, 60)
button_hoch = Rect(220, 480, 80, 60)
button_runter = Rect(300, 480, 80, 60)

speed = 15

def draw():
    screen.fill("darkblue")

    # Spielbereich
    screen.draw.rect(Rect(10, 10, WIDTH - 20, 450), "white")

    # Spieler
    screen.draw.filled_rect(Rect(spieler_x, spieler_y, 40, 40), "red")

    # Buttons zeichnen
    screen.draw.filled_rect(button_links, "gray")
    screen.draw.filled_rect(button_rechts, "gray")
    screen.draw.filled_rect(button_hoch, "gray")
    screen.draw.filled_rect(button_runter, "gray")

    # Button Rahmen
    screen.draw.rect(button_links, "white")
    screen.draw.rect(button_rechts, "white")
    screen.draw.rect(button_hoch, "white")
    screen.draw.rect(button_runter, "white")

    # Button Beschriftungen
    screen.draw.text("←", (45, 495), color="white", fontsize=35)
    screen.draw.text("→", (145, 495), color="white", fontsize=35)
    screen.draw.text("↑", (248, 495), color="white", fontsize=35)
    screen.draw.text("↓", (328, 495), color="white", fontsize=35)

def on_mouse_down(pos):
    global spieler_x, spieler_y

    # Prüfen welcher Button gedrückt wurde
```

```
    # collidepoint(pos) gibt True zurück wenn pos im Rechteck ist
    if button_links.collidepoint(pos):
        spieler_x = spieler_x - speed

    if button_rechts.collidepoint(pos):
        spieler_x = spieler_x + speed

    if button_hoch.collidepoint(pos):
        spieler_y = spieler_y - speed

    if button_runter.collidepoint(pos):
        spieler_y = spieler_y + speed

    # Grenzen prüfen
    if spieler_x < 15:
        spieler_x = 15
    if spieler_x > WIDTH - 55:
        spieler_x = WIDTH - 55
    if spieler_y < 15:
        spieler_y = 15
    if spieler_y > 410:
        spieler_y = 410
```

Zusammenfassung: Was du gelernt hast

Die wichtigsten Funktionen

Funktion	Wann wird sie aufgerufen?
`draw()`	Um den Bildschirm zu zeichnen
`update()`	60 mal pro Sekunde (für Bewegung)
`on_mouse_down(pos)`	Wenn du auf den Bildschirm tippst
`on_mouse_move(pos)`	Wenn du den Finger über den Bildschirm ziehst

Die wichtigsten Zeichenbefehle

Befehl	Was es macht
`screen.fill("farbe")`	Hintergrund füllen
`screen.draw.filled_rect(Rect(...), "farbe")`	Rechteck zeichnen
`screen.draw.filled_circle((x,y), radius, "farbe")`	Kreis zeichnen
`screen.draw.text("text", (x,y), color="farbe")`	Text schreiben
`screen.draw.line((x1,y1), (x2,y2), "farbe")`	Linie zeichnen

Wichtige Konzepte

1. **Koordinaten:** x = links/rechts, y = oben/unten (y=0 ist OBEN!)
2. **global:** Brauchst du um Variablen in Funktionen zu ändern

3. **Kollision:** Prüfen ob sich zwei Rechtecke überlappen
4. **Schwerkraft:** Geschwindigkeit wird immer größer → Objekt fällt schneller
5. **Listen:** Speichern viele Objekte zusammen (z.B. alle Steine)

Häufige Fehler und Lösungen

Fehler	Lösung
`NameError: name 'punkte' is not defined`	Hast du `global punkte` vergessen?
Nichts passiert beim Tippen	Prüfe ob `on_mouse_down(pos)` richtig geschrieben ist
Das Spiel startet nicht	Hast du die Datei mit `.py` gespeichert?
Der Ball geht durch den Schläger	Prüfe deine Kollisionsabfrage
Variablen ändern sich nicht	Hast du alle nötigen Variablen als `global` markiert?

Nächste Schritte

1. **Probiere die Spiele aus** und ändere die Werte (Geschwindigkeit, Größen, Farben)
2. **Kombiniere Ideen** aus verschiedenen Spielen
3. **Lies Teil 2** um zu lernen wie man Bilder und Sounds hinzufügt!

Viel Spaß beim Programmieren! 🎮