



2D-Spiele mit pygame

Prof. Dr. Oliver Hofmann

Worum geht es?

Mit Hilfe des Pakets

pygame

lassen sich in Python recht
einfach 2D-Spiele
programmieren.

Das Paket ist über den
Python Package Index (PyPI)
zu beziehen.

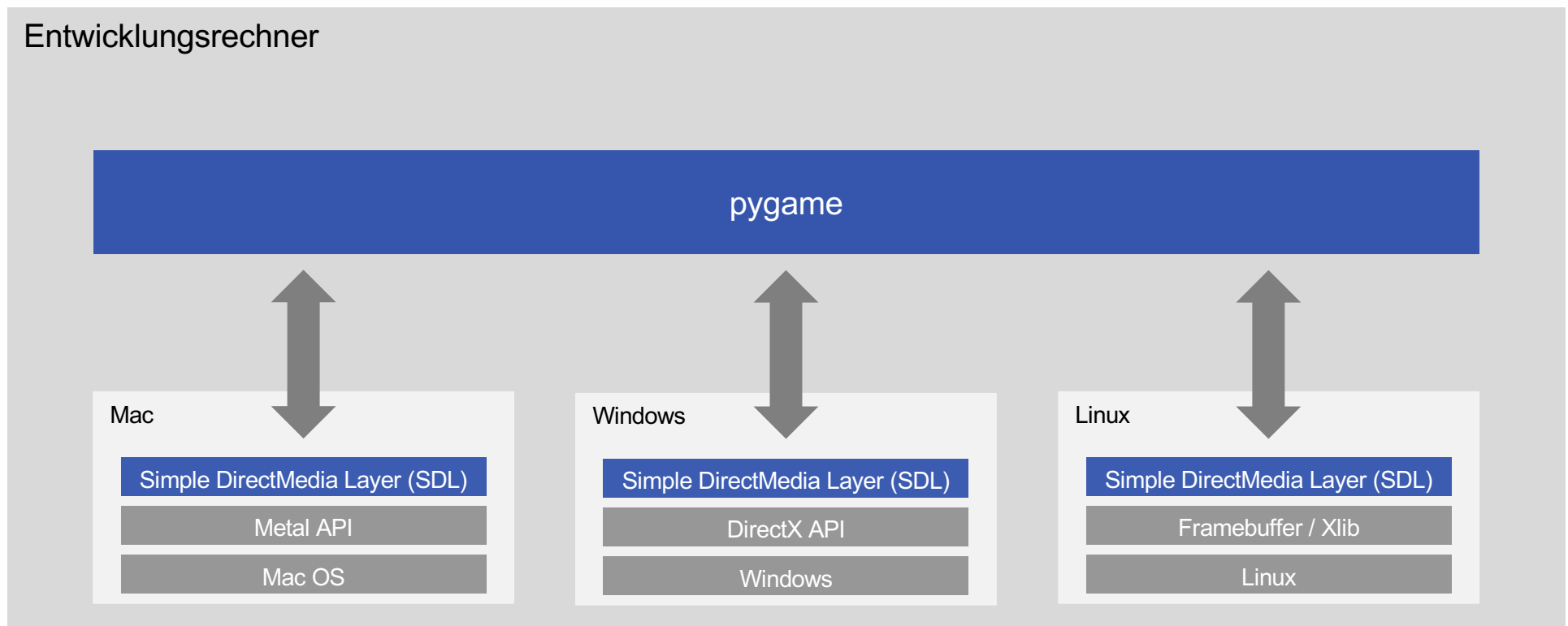
Tutorials und Dokumentation
findet sich bei

www.pygame.org



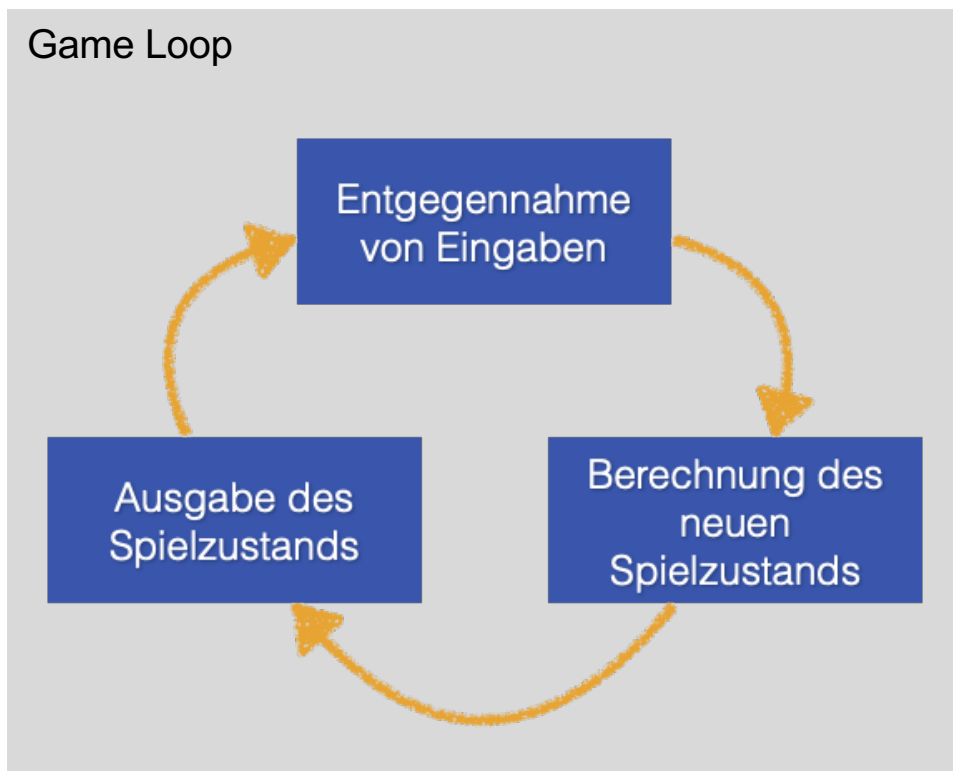
Worum geht es?

Die Installation mit **Pip** kann etwas hakelig sein, da auch systemspezifische Bibliotheken zur Grafikprogrammierung installiert bzw. genutzt werden.



Game Loop

Kern jedes Spiels ist meist eine Endlosschleife, in der in schneller Folge immer wieder die gleichen Aufgaben abgearbeitet werden:



```
import pygame

def init():...
def draw():...
def update():...
def user_input():...

init()

running = True

while running:
    running = user_input()
    update()
    draw()
```

Initialisierung des Spielzustands beim Start

Game Loop

Initialisierung

Beim Spielstart müssen

- die beteiligten Bibliotheken initialisiert werden
- die Oberfläche vorbereitet werden
- die Variablen, in denen das Spielgeschehen festgehalten wird, mit Anfangswerten versorgt werden

Bei kleinen Spielen kann das in einer Funktion **init** erfolgen, bei größeren Spielen wird die Funktionalität auf mehrere Funktionen und Objekte verteilt.

Bei größeren Mengen an Spieldaten ggf. Dictionary oder Module bzw. Objekte verwenden

```
def init():  
    global object_position  
    global object_size, object_speed  
  
    # Initialisierung der Bibliothek pygame  
    pygame.init()  
  
    # Vorbereiten der Oberfläche  
    size = (320, 240)  
    pygame.display.set_mode(size)  
    pygame.display.set_caption("Hello World")  
  
    # Spieldaten initialisieren  
    object_position = [size[0]//2, size[1]//2]  
    object_speed = [2, 2]  
    object_size = (50, 50)
```

Spieldaten als globale Variable

Ausgabe des Spielzustands

Bei der Darstellung des Spielzustands erfolgt lediglich lesender Zugriff auf die Spieldaten.

Alle Veränderungen werden zunächst unsichtbar vorbereitet. Erst mit

pygame.display.flip

wird die aktuelle Anzeige durch die vorbereitete Anzeige ausgetauscht.

Zeichnen eines Rechtecks an den von den Spieldaten vorgegeben Position

```
def draw():
    # Benötigte Farben als RGB-Werte
    black = (0, 0, 0)
    red = (255, 0, 0)

    # Zeichnen des aktuellen Spielzustands
    window = pygame.display.get_surface()
    window.fill(black)
    rect = pygame.Rect(object_position[0],
                        object_position[1],
                        object_size[0],
                        object_size[1])
    pygame.draw.rect(window, red, rect)

    # "Umschalten" der aktuellen Anzeige auf die
    # gerade gezeichnete Anzeige
    pygame.display.flip()
```

Berechnung des neuen Spielzustands

Bei jedem Durchlauf der Game Loop muss der Spielzustand neu berechnet werden, da

- in der vergangenen Zeit Objekte ihren Zustand (z.B. Position, Farbe) geändert haben können
- Ereignisse der Spiellogik (z.B. Kollisionen, Aktionen der Spielgegner) stattgefunden haben können
- Eingaben des Spielers zu Reaktionen führen können

```
def update():  
    global object_position  
    global object_speed  
  
    # Abmessungen des Game-Fensters ermitteln  
    window = pygame.display.get_surface()  
    window_size = (window.get_width(),  
                   window.get_height())  
  
    for dim in [0, 1]:  
        # Bewegung des Objekts  
        object_position[dim] += object_speed[dim]  
        # Bouncing?  
        if object_position[dim] < 0:  
            object_speed[dim] *= -1  
        end = object_position[dim] + object_size[dim]  
        if end > window_size[dim]:  
            object_speed[dim] *= -1
```

Entgegennahme von Eingaben

Eingaben des Spielers werden in einer Warteschlange gepuffert.

Bei jedem Durchlauf der Game Loop sollte diese Warteschlange komplett abgearbeitet werden.

Pygame unterscheidet z.B.

- pygame.QUIT
Fenster wurde geschlossen
- pygame.KEYDOWN
Taste wurde gedrückt
- pygame.KEYUP
Taste wurde losgelassen

```
def user_input():  
  
    # Alle Events seit dem letzten Durchlauf  
    for event in pygame.event.get():  
        if event.type == pygame.QUIT:  
            return False  
        if event.type == pygame.KEYDOWN:  
            if event.key == pygame.K_ESCAPE:  
                return False  
    return True
```


Ausgabe von Bildern

Bilder der Formate **png**, **jpg** oder **gif** können in der Game Loop ebenfalls ausgegeben werden.

Die Bilddatei sollte bereits vorab eingelesen werden, um den Zeitbedarf in der Loop zu minimieren.

Die Ausgabe des Bildes erfolgt dann in mittels

window.blit

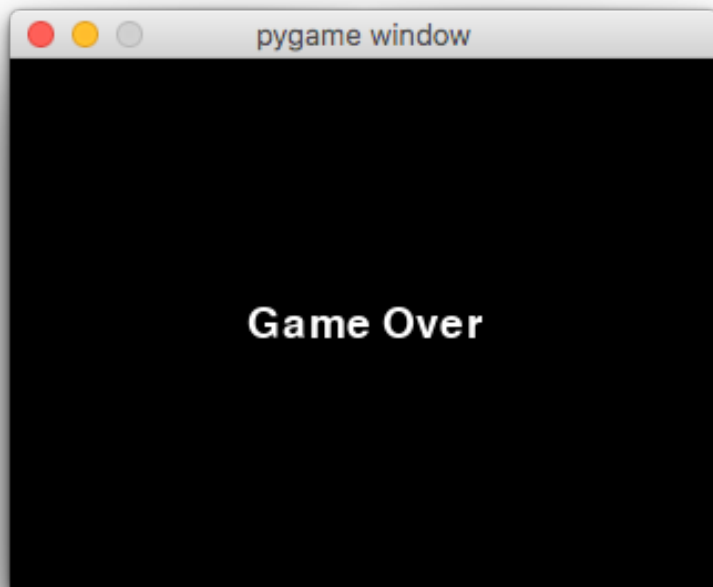
wodurch das Bild an der angegebenen Position in die Anzeige eingefügt wird.

```
def init():  
    global object_image  
    object_image = pygame.image.load('ohm.png')
```

```
def draw():  
    # Benötigte Farben als RGB-Werte  
    black = (0, 0, 0)  
  
    # Zeichnen des aktuellen Spielzustands  
    window = pygame.display.get_surface()  
    window.fill(black)  
    window.blit(object_image, object_position)  
  
    # "Umschalten" der aktuellen Anzeige auf die  
    # gerade gezeichnete Anzeige  
    pygame.display.flip()
```

Ausgabe von Text

Text kann in **pygame** als Bild gerendert werden. Die Ausgabe erfolgt dann analog zur Bildausgabe.



```
def init():  
    global text_image  
  
    # ...  
  
    white = (255, 255, 255)  
    font = pygame.font.Font(None, 28)  
    text_image = font.render('Game Over', 1, white)
```

Zusammenfassung

- Mit dem Paket **pygame** können 2D-Spiele entwickelt werden
- Eine wichtige Kontrollstruktur ist die **Game Loop**, die im Spiel wiederholt durchlaufen werden muss
- In der Game Loop werden
 - alle Eingaben verarbeitet
 - der Spielzustand berechnet
 - der Spielzustand ausgegeben