Ein Spiel programmieren mit Pygame Zero

In dieser Aufgabenserie lernst du ein Spiel mit Pygame Zero zu erstellen. Du kannst die Aufgaben allein oder zusammen mit deiner Nachbarin lösen. Löse die Aufgaben der Reihe nach und versuche sie aufmerksam durchzulesen und zu verstehen. **Falls du etwas nicht verstehst oder bei einer Aufgabe stecken bleibst, kannst du jederzeit eine Tutorin um Hilfe fragen.**

# Mu Editor

Um in Python programmieren zu können brauchst du einen **Editor** oder eine Entwicklungsumgebung, in der du **deinen Code schreiben** kannst. Im Workshop verwenden wir den **Mu Editor**. Falls du zuhause gerne weiter coden möchtest, kannst du Mu auch ganz einfach auf deinem Computer installieren (frage vorher deine Eltern um Erlaubnis). Den Download für den Mu Editor findest du hier: <https://codewith.mu/en/download>

Mu ist kostenlos und Python 3 sowie Pygame Zero sind bereits vorinstalliert.

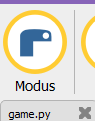
# Den Mu Editor verwenden

Wenn du Mu zum ersten Mal startest, kannst du auswählen welche Art von Programm du schreiben möchtest. Für ein normales Python Programm wähle Python 3. Da wir im Workshop ein Spiel mit **Pygame Zero** entwickeln wollen, wähle Pygame Zero aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Du kannst diese Einstellung jederzeit ändern, indem du oben links auf "Modus" klickst.



Jetzt kannst du anfangen deinen Code zu schreiben.

Alles was mit einem **"#"** beginnt ist ein **Kommentar** und gehört **nicht** zum Code. Du kannst diese Zeile löschen oder ignorieren. Kommentare können nützlich sein, um dich später daran zu erinnern, wozu ein gewisser Teil des Codes dient. Du kannst jederzeit eigene Kommentare schreiben, indem du eine Zeile mit "#" beginnst.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

# Ein Spiel mit Pygame Zero

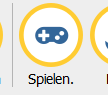
## Aufgabe 1: Das Spielfenster

Da du nun den Mu Editor etwas kennengelernt hast, können wir anfangen ein Spiel zu programmieren.

Als erstes brauchen wir ein **Fenster**, in dem das Spiel später erscheinen wird. Dafür müssen wir die Höhe (Englisch height) und die Breite (Englisch width) des Fensters angeben. Kopiere den folgenden Code in deinen Mu Editor und klicke auf “Spielen”, um den Code auszuführen.

WIDTH = 600

HEIGHT = 600



Mit "Stopp" kannst du das Programm wieder beenden.

❔ Probiere nun die beiden Zahlen zu verändern und schaue was passiert.

**WIDTH** und **HEIGHT** sind sogenannte Konstanten. Sie bleiben während der ganzen Zeit gleich, in der das Programm ausgeführt wird. Namen von Konstanten werden in Python immer in Grossbuchstaben geschrieben.

Für unser Spiel werden wir eine **Fenstergösse von 700 x 700** **Pixel** verwenden, setzte WIDTH und HEIGHT also beide auf 700.

## Aufgabe 2: Der Hase, unsere Spielfigur

Da wir jetzt ein Fenster haben, wollen wir unsere Spielfigur, den Hasen, einfügen. Dazu musst du den folgenden Code hinzufügen:

hase = Actor('bunny1\_stand')

hase.x = 350

hase.y = 350

In der 1. Zeile dieses Codes erstellen wir unsere **Spielfigur**. Diese ist ein **„Actor“** (von Englisch „action“, also jemand der etwas tut). In Klammern geben wir den **Namen des Bildes** an, welchen wir verwenden wollen. Dabei ist es wichtig, dass sich das Bild **im images Ordner** befindet und der Name in Anführungszeichen geschrieben wird. Wir geben unserer Spielfigur den Namen **„hase“** damit wir sie später wieder ansprechen können.

Mit **hase.x** und **hase.y** kannst du die **Position** der Figur im Spielfenster anpassen. Wir wollen den Hasen in der Mitte platzieren. Da unser Spielfenster 700 x 700 Pixel gross ist, können wir hase.x und hase.y beide auf 350 setzen (die Hälfte von 700).

Im Gegensatz zu Konstanten wie WIDTH oder HEIGHT nennt man hase.x und hase.y **Variablen**. Sie werden normalerweise **im Laufe des Programms verändert**, wie hier zum Beispiel die Position des Hasen.

Kopiere jetzt noch den folgenden Code ganz ans Ende deines Programms:

def draw():

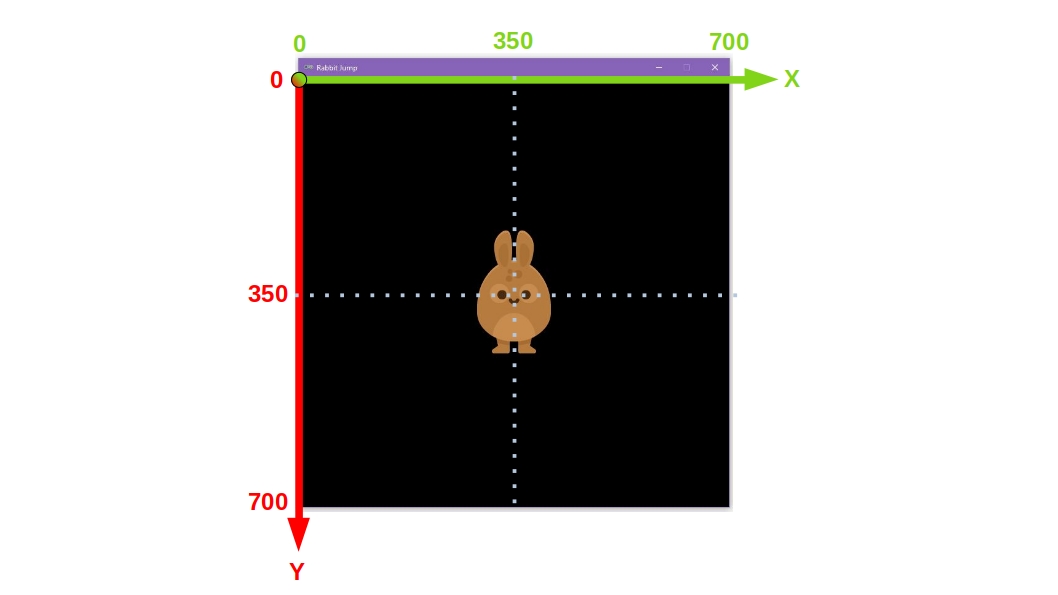
screen.clear()

hase.draw()

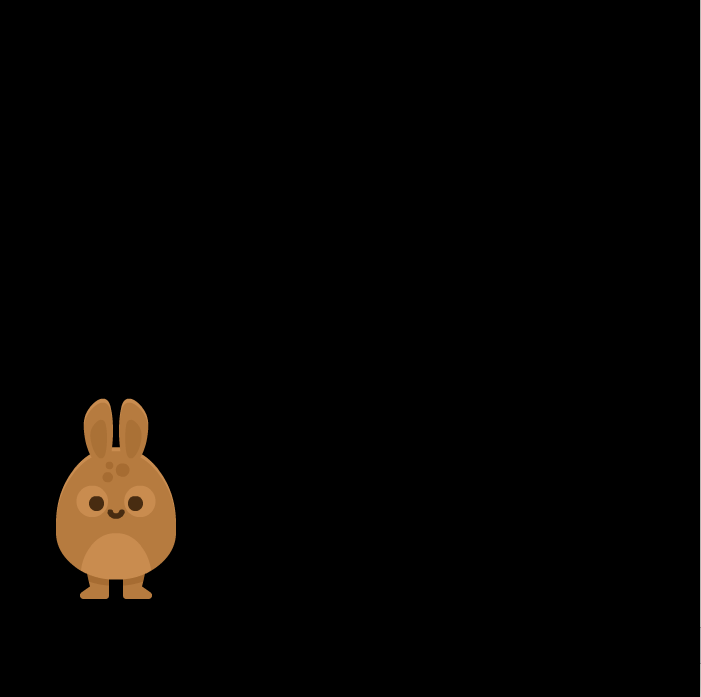
Das ist eine Funktion, die dafür verantwortlich ist, etwas im Fenster zu zeichnen („draw“ heisst auf Englisch zeichnen). Zum Thema Funktionen lernst du mehr in der nächsten Aufgabe.

Jetzt kannst du auf „Spielen“ klicken und der Hase sollte in der Mitte des Bildschirms erscheinen. Klicke auf „Stopp“ und schaue was passiert, wenn du die Zahlen von hase.x und hase.y veränderst.

Schau dir das folgende Bild an um zu verstehen, wie Dinge im Fenster platziert werden können:

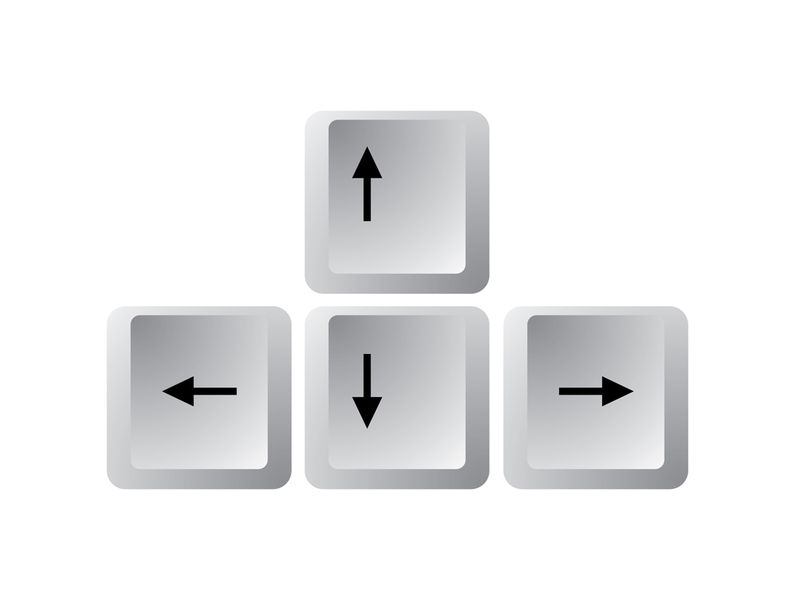


❔ Versuche jetzt den Hasen ungefähr so in der unteren linken Ecke zu platzieren, dass er auf einer Höhe von y=500 steht:



## Aufgabe 3: Den Hasen bewegen

Damit sich unser Hase bewegen kann, benutzen wir **die „update“ Funktion**. Eine Funktion ist ein Teil eines Programms oder ein **Unterprogramm**. Funktionen helfen uns ein grosses Programm **aufzuteilen** und übersichtlich zu machen.

Als erstes wollen wir den Hasen nach links und rechts laufen lassen, wenn wir die **Pfeiltasten** für links oder rechts drücken. 

In Pygame Zero (und anderen Game Engines) hat gibt es für **jede Taste auf der Tastatur oder Maus einen Namen**. Die linke Pfeiltaste heisst LEFT und die rechte Pfeiltaste heisst RIGHT. Mit **keys.LEFT** und **keys.RIGHT** können wir die Tasten ansprechen (Keys heisst auf Englisch Tasten).

Füge den folgenden Code ganz am Ende ein:

def update():

if keyboard[keys.RIGHT]:

hase.x = hase.x + 1

if keyboard[keys.LEFT]:

hase.x = hase.x - 1

In diesem Code Stück lernst du ein weiteres wichtiges Element des Programmierens kennen, das **if-statement**. Das wort **“if”** bedeutet auf Englisch so viel wie **“wenn”** oder **“falls”**. Damit wollen wir signalisieren, dass wir etwas tun wollen, **nur falls etwas bestimmtes zutrifft**. Schaue dir den folgenden Ausschnitt auf den Code an:

if keyboard[keys.RIGHT]:

hase.x = hase.x + 1

❔ Kannst du dir vorstellen was hier passiert?

**Tipp:** Keyboard heisst auf Englisch Tastatur (falls du nicht mehr weisst was hase.x bedeutet schau nochmals bei der letzten Aufgabe nach)

Hast du herausgefunden was dieser Code tut? Genau, der Hase **bewegt sich nach rechts** (hase.x + 1), immer wenn die rechte Pfeiltaste gedrückt wird. Der zweite teil des Coeds tut das gleiche für die linke seite (hase.x - 1) wenn die linke Pfeiltaste gedrückt wird.

Klicke jetzt auf “Spielen” und schau, ob du den Hasen bewegen kannst.

❔ Der Hase bewegt sich noch etwas langsam. Versuche jetzt herauszufinden, wie du ihn schneller machen kannst.

**Tipp:** Du musst nur 2 bestimmte Zahlen anpassen, um die Geschwindigkeit des Hasen zu ändern.

Aufgabe 4: Ein hüpfender Hase

Der Hase kann jetzt nach links und rechts gehen, aber natürlich möchte er auch hüpfen können, wie Hasen das ebenso machen.

Wir wollen den Hasen hüpfen lassen, wenn wir die Leertaste (auch Space genannt) drücken.

Wenn der Hase springt, muss er danach natürlich auch wieder zurück auf dem Boden landen.

Füge zunächst den folgenden Code zuunterst zu deiner „update“ Funktion hinzu:

if hase.y < 500:

hase.y = hase.y + 2

Dieser Code sorgt dafür, dass der Hase wider **zurück auf den Boden** kommt, wenn hase.y kleiner als 500 ist (also weiter oben als die Startposition).

Natürlich kann der Hase auch nicht unendlich hoch springen. Um das zu verhindern kannst du einen Boolean nutzen. Ein **Boolean ist eine spezielle Art** **von Variablen** die nur **zwei Zustände** haben kann, **True** (wahr) oder **False** (falsch). Da wir diese Variable immer wieder brauchen werden, ist es wichtig diese Zeile ganz oben im Code (z.B. nach WIDTH und HEIGHT), ausserhalb der Funktionen hinzuzufügen:

jump\_down = False

Jetzt kannst du die “update“ Funktion ergänzen:

if keyboard[keys.SPACE]:

global jump\_down

if jump\_down and hase.y < 500:

return

elif jump\_down and hase.y >= 500:

jump\_down = False

elif hase.y >= 200 and not jump\_down:

hase.y = hase.y - 5

elif hase.y < 200:

jump\_down = True

Hier wird wieder ein if-statement gebraucht. Mit **„elif“** können wir ein if-statement ergänzen, wenn aus mehreren möglichen Situationen prüfen möchten, ob eine zutrifft.

❔ Starte das Spiel und versuche den Hasen hüpfen zu lassen. Fällt dir etwas auf?

Genau! Jetzt müssen wir noch dafür sorgen, dass der Hase nicht in der Luft bleibt solange wir die Space Taste immer wieder drücken und aus der Luft nach oben springt.

Du kannst die folgende Funktion nach der „update“ Funktion im Code hinzufügen:

def on\_key\_up(key):

if key == keys.SPACE:

global jump\_down

jump\_down = True

❔Schaue dir die Funktion noch etwas genauer an kannst du dir vorstellen was on\_key\_up bedeutet und was die Funktion in etwa macht?

Super! Jetzt kann der Hase endlich hüpfen!

Aufgabe 5: Eine schöne Landschaft

Da ein schwarzer Hintergrund nicht sehr spannend ist, kannst du ganz einfach ein Bild als Hintergrund einfügen.

Füge dazu einfach diese Codezeile irgendwo in der „draw“ Funktion ein:

screen.blit('background', (0, 0))

Diese fügt im Fenster das Bild mit dem Namen „background“ als Hintergrund ein.

Aufgabe 6: Karotten und Monster

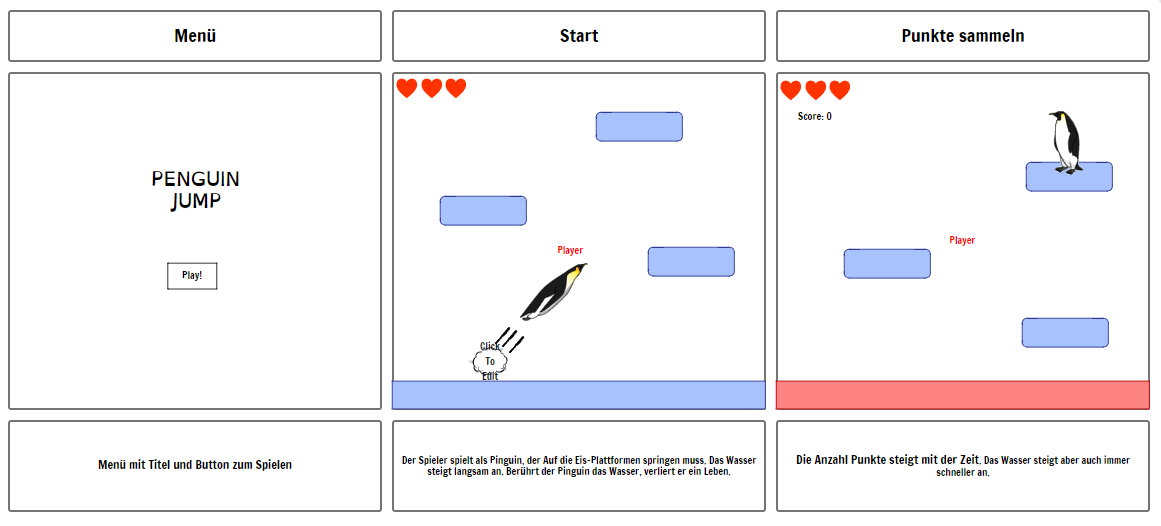
Damit unser Hase auch etwas zu tun hat, wollen wir Karotten hinzufügen, die durch die Luft fliegen und die der Hase einsammeln muss.

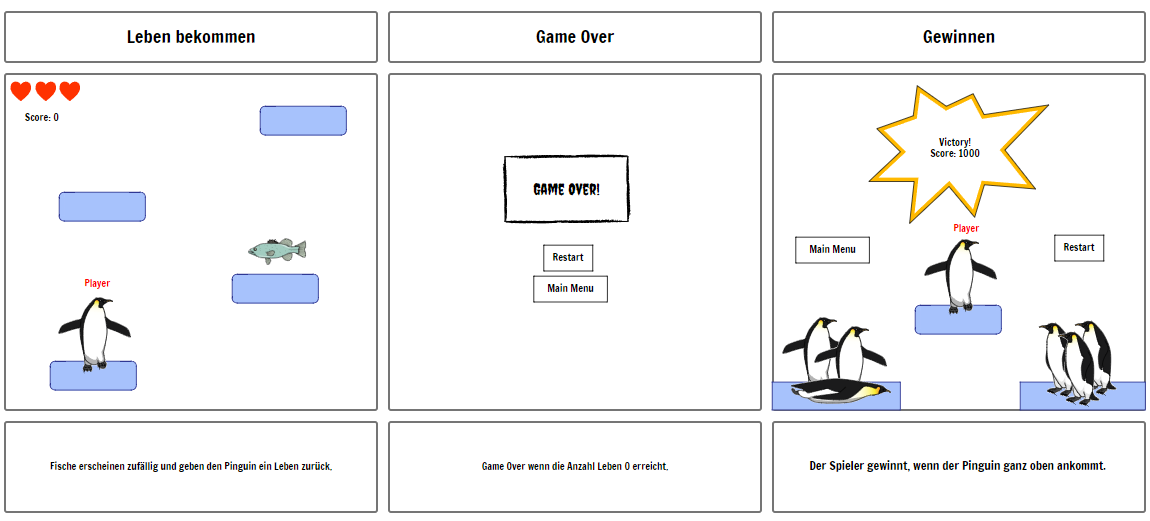
Aufgabe 7: Punkte und Game Over

.

Zusatzaufgabe

Werde jetzt selbst kreativ! Game Developer benutzen oft Storyboards um Ideen für Spiele darzustellen. Ein Storyboard könnte zum Beispiel so Aussehen:





Eine Vorlage für ein Storyboard findest du auf der nächsten Seite.

Lösungen und Unterlagen

Alle Lösungen und Unterlagen zum Workshop findest du auf Dropbox: <https://www.dropbox.com/sh/trq796ttod52yxv/AAC1RtvkKB8GgxaBlr4bQWa_a?dl=0>