

Philipps



Universität  
Marburg

# Game Programming: From Scratch to Python

Dienstag

30.08.2022

9:30 – 11:30am



hessian.AI

# Python Review



# Review question

What are the **datatypes** (types of variables) of var1, var2, and var3?

```
var1 = "Game Over : ("
var2 = 200
var3 = 20.0
```

# Review exercise

Finish the for-loop such that it print the value if it is an even number and less than 10, or if it is an odd number and greater than 10.

Rules:

1. Include at least one **if** and one **elif**
2. Use the **modulo** operator
3. Use **compound operators**

```
for i in range(1,21):  
    """ your code below """
```

# Review solution

Finish the for-loop such that it print the value if it is an even number and less than 10, or if it is an odd number and greater than 10.

```
for i in range(1,21):  
    if i < 10 and i % 2 == 0:  
        print(i)  
    elif i >= 10 and i % 2 == 1:  
        print(i)
```

# Review: Spot the problem!

What's wrong with this code?

```
my_print_function(message):  
    print("Message:", message)  
  
my_print_function("Hello!")
```

# Review: Spot the problem!

What's wrong with this code?

`def` was missing before the function name in the function definition

```
def my_print_function(message):  
    print("Message:", message)  
  
my_print_function("Hello!")
```

# Review: Spot the problem!

What's wrong with this code?

```
score = 0
lives = 3
while lives != 0:
    if banana_position == ground_position:
        go_to_x(banana_position)
    if banana_position == player_position:
        go_to_x(banana_position)
        score = score + 1
```



# Review: Spot the problem!

What's wrong with this code?

The **stopping condition** will never be reached causing an **infinite loop**

We need to subtract 1 from lives when the banana hits the ground

```
lives = lives - 1
```

```
score = 0
lives = 3
while lives != 0:
    if banana_position == ground_position:
        go_to_x(banana_position)
        lives = lives - 1
    if banana_position == player_position:
        go_to_x(banana_position)
        score = score + 1
```

# Review: True or False?

```
materials = ["clay", "wood", "stone"]  
materials[1] == "clay"
```



# Review: True or False?

```
materials = ["clay", "wood", "stone"]  
materials[1] == "clay"
```

**False**



# Review: True or False?

```
materials = ["clay", "wood", "stone"]  
materials[1][2] == materials[-1][2]
```



# Review: True or False?

```
materials = ["clay", "wood", "stone"]  
materials[1][2] == materials[-1][2]
```

**True**

materials[1] == "wood"  
materials[-1] == "stone"

# Review: True or False?

The code on the left and the code on the right do the same thing.

```
i = 1
while i < 21:
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
    i = i + 1
```

```
for i in range(1,21):
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
```

# Review: True or False?

The code on the left and the code on the right do the same thing.

```
i = 1
while i < 21:
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
    i = i + 1
```

```
for i in range(1,21):
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
```

**True**

# Review: True or False?

The code on the left and the code on the right do the same thing.

```
i = 1
while True:
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)

    if i == 21:
        break
    i = i + 1
```

```
for i in range(1,21):
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
```



# Review: True or False?

The code on the left and the code on the right do the same thing.

```
i = 1
while True:
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)

    if i == 21:
        break
    i = i + 1
```

```
for i in range(1,21):
    if i < 10 and i % 2 == 0:
        print(i)
    elif i >= 10 and i % 2 == 1:
        print(i)
```

**False: the code on the left will print 21**

# Dictionaries review

Complete the for-loop to make the dictionary map the item of the list to the item's index in the list (keys are items and values are indices).

The enumerate function can tell you the iteration (index) you are currently on. That value is stored in the index variable, and the list item is stored in the item variable.

```
grocery_list = ["Onions", "Tomatoes", "Rice", "Soup"]  
item_to_index = {}  
  
for index, item in enumerate(grocery_list):  
    """ complete the for-loop """
```

# Dictionaries review

```
grocery_list = ["Onions", "Tomatoes", "Rice", "Soup"]  
item_to_index = {}  
  
for index, item in enumerate(grocery_list):  
    item_to_index[item] = index  
  
print(item_to_index['Onions'], item_to_index["Soup"])
```

# A Soft Introduction to Pygame

...and game programming





Pygame Rects and Pixel Coordinates

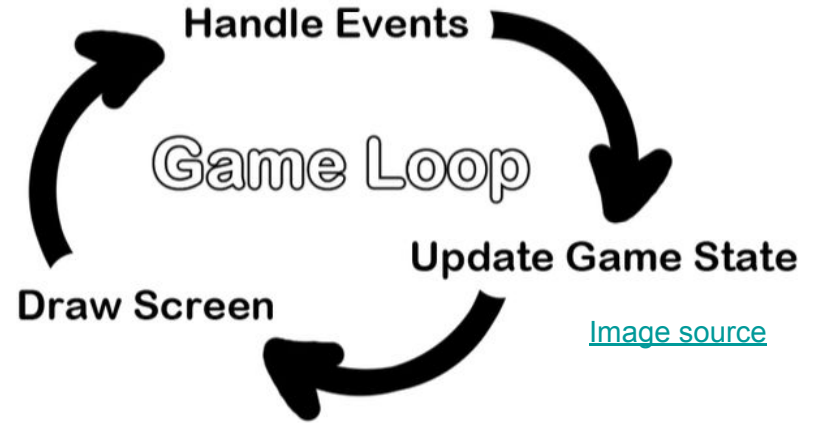
# Part 1

# Main game loop

Die “Game Loop” ist das Herz eines Videospiels.

Drei wichtige Dinge passieren in der Game Loop:

1. **Handle Events** (wie das Drücken einer Taste oder das Klicken der Maus)
2. **Update Game Status** (z. B. wo sich Pacman und die Geister befinden)
3. **Draw Screen** (macht das Spiel im Fenster sichtbar)



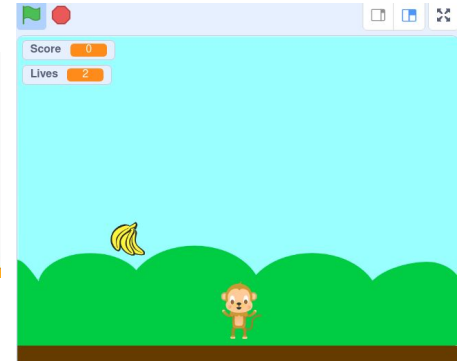
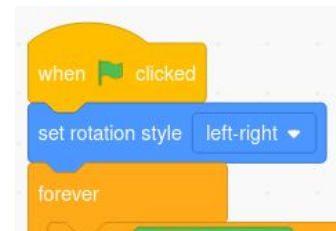
# Main game loop

Die “Game Loop” ist das Herz eines Videospiels.

Drei wichtige Dinge passieren in der Game Loop:

1. **Handle Events** (wie das Drücken einer Taste oder das Klicken der Maus)
2. **Update Game Status** (z. B. wo sich Pacman und die Geister befinden)
3. **Draw Screen** (macht das Spiel im Fenster sichtbar)

Wenn Du in Scratch auf die grüne Flagge klickst, beginnt die Game Loop.



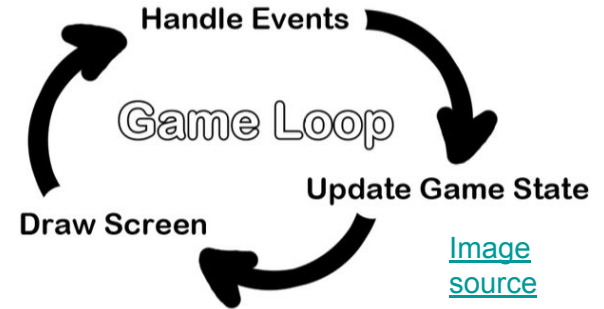
# Main game loop: Pygame event Beispiel



<https://www.pygame.org/docs/>

**Pygame** ist eine Python-Bibliothek mit Funktionen und Objekten, die uns helfen, die drei großen Aufgaben der Spielschleife zu erfüllen.

Hier verwenden wir Pygame, um herauszufinden, ob die Schaltfläche zum Verlassen des Fensters angeklickt wurde, damit wir wissen, dass wir den Code zum Verlassen des Spiels und Beenden des Programms ausführen müssen.



```
while True: # main game loop
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
```



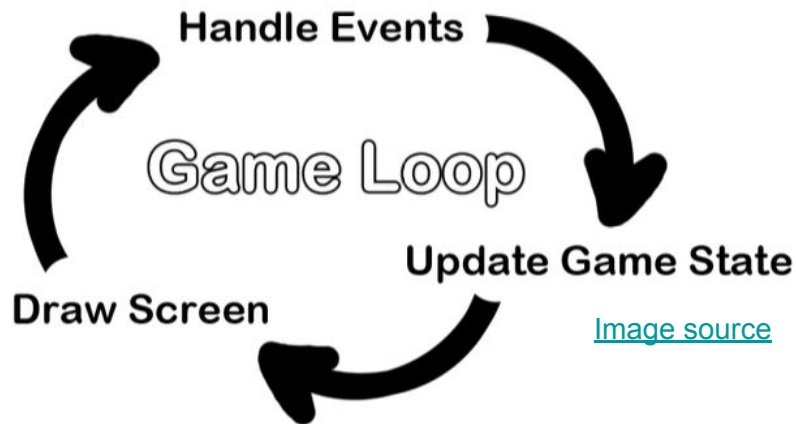
# Main game loop

Die “Game Loop” ist das Herz eines Videospiels.

Drei wichtige Dinge passieren in der Game Loop:

1. **Handle Events** (wie das Drücken einer Taste oder das Klicken der Maus)
2. **Update Game Status** (z. B. wo sich Pacman und die Geister befinden)
3. **Draw Screen** (macht das Spiel im Fenster sichtbar)

Eine Iteration der Game Loop ist ein *Frame*

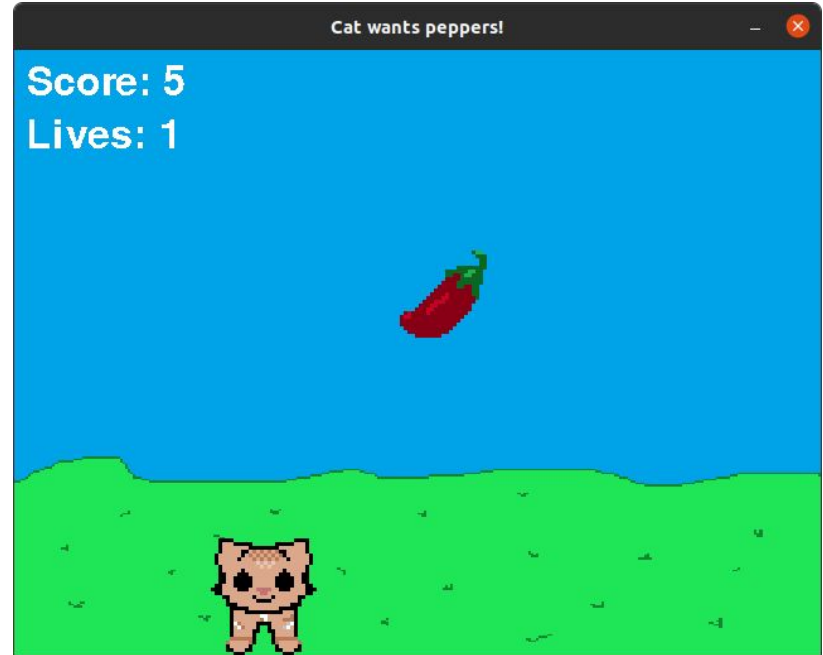


**Mehr zu diesen drei Aufgaben und Schleifeniterationen später...**

# Pixelkoordinaten: Wo die Dinge auf dem Bildschirm sind

In Pygame verläuft die y-Achse von oben nach unten und die x-Achse von links nach rechts.

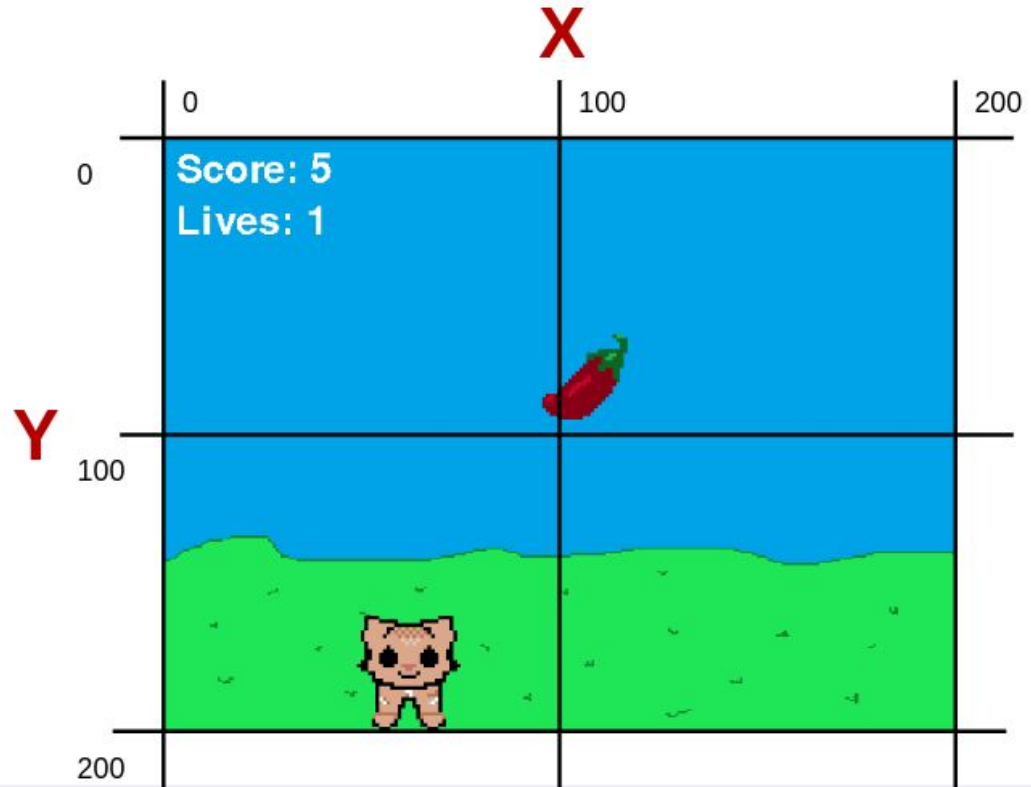
	X							
	0	1	2	3	4	5	6	7
0					■			
1								
2			■					
3								
4			■					
5	■							
6						■		
7								



# Pixelkoordinaten: Wo die Dinge auf dem Bildschirm sind

Wo sind die folgenden Objekte (ungefähr)?

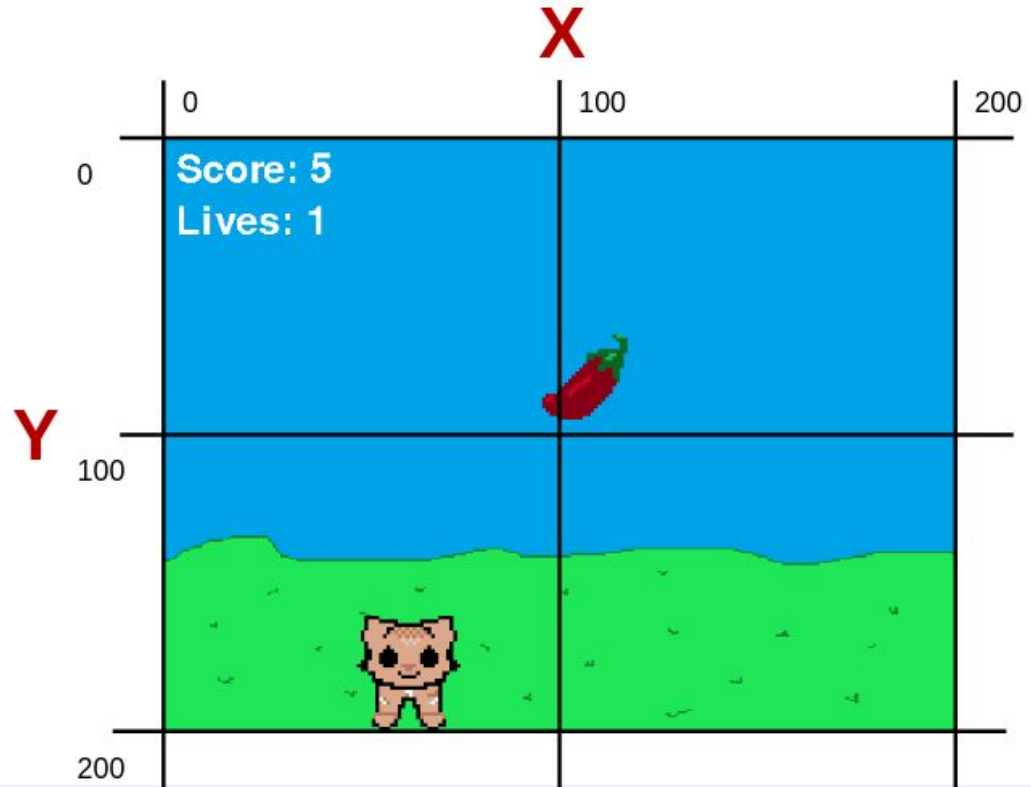
1. Score: 5  
( $0 \leq x < 50$ ,  $0 \leq y < 20$ )
2. Lives : 1
3. Cat:
4. Pepper



# Pixelkoordinaten: Wo die Dinge auf dem Bildschirm sind

Wo sind die folgenden Objekte (ungefähr)?

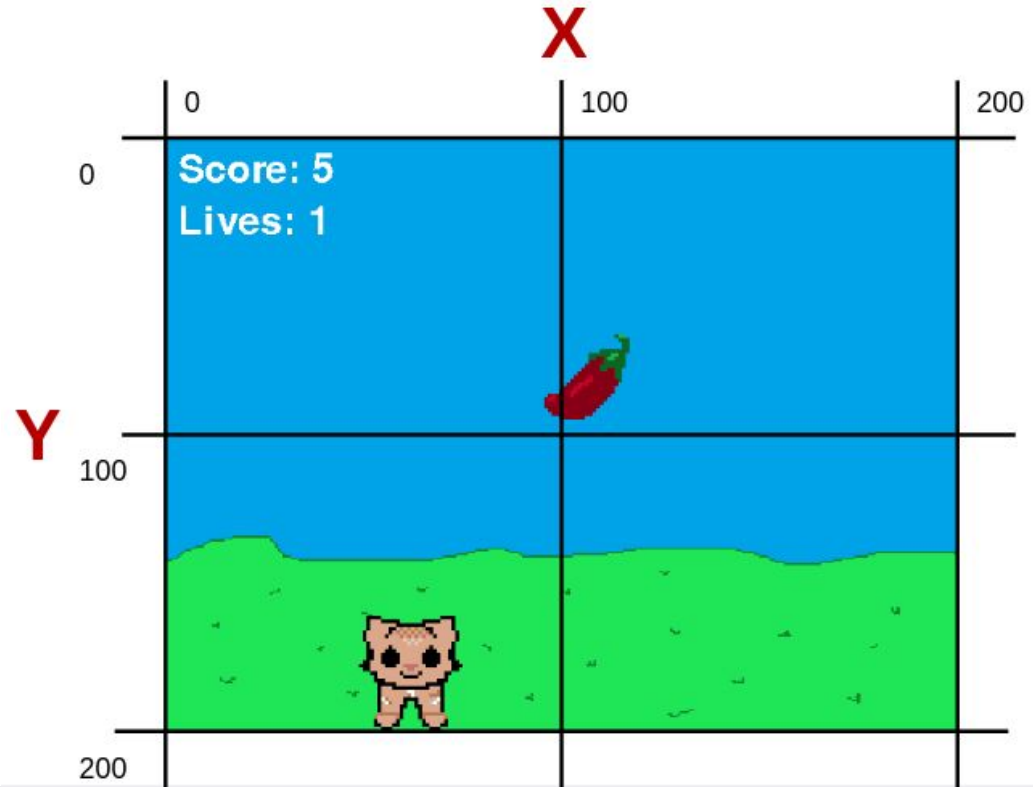
1. Score: 5  
( $0 \leq x < 50$ ,  $0 \leq y < 20$ )
2. Lives : 1  
( $0 \leq x < 50$ ,  $10 \leq y < 40$ )
3. Cat:
4. Pepper



# Pixelkoordinaten: Wo die Dinge auf dem Bildschirm sind

Wo sind die folgenden Objekte (ungefähr)?

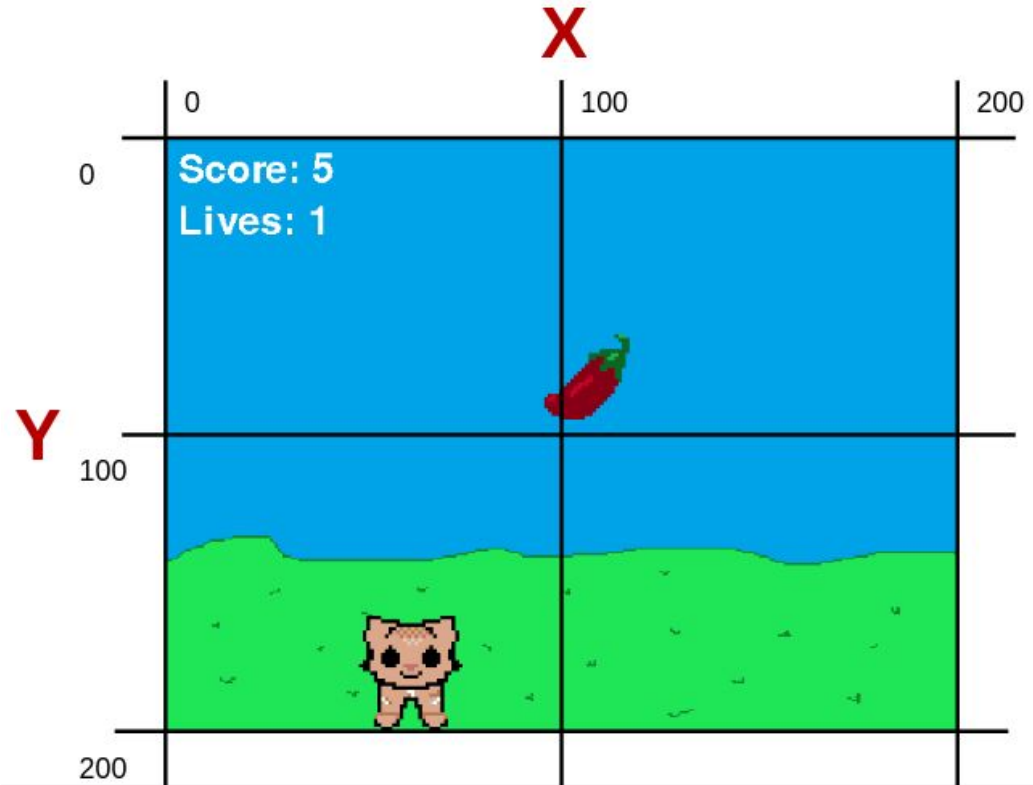
1. Score: 5  
( $0 \leq x < 50$ ,  $0 \leq y < 20$ )
2. Lives : 1  
( $0 \leq x < 50$ ,  $10 \leq y < 40$ )
3. Cat  
( $50 \leq x < 80$ ,  $150 < y < 200$ )
4. Pepper



# Pixelkoordinaten: Wo die Dinge auf dem Bildschirm sind

Wo sind die folgenden Objekte (ungefähr)?

1. Score: 5  
( $0 \leq x < 50$ ,  $0 \leq y < 20$ )
2. Lives : 1  
( $0 \leq x < 50$ ,  $10 \leq y < 40$ )
3. Cat  
( $50 \leq x < 80$ ,  $150 < y < 200$ )
4. Pepper  
( $95 < x \leq 110$ ,  $60 < y < 99$ )



# Sprites, Surfaces, and Rectangles

Das ist ein Sprite



# Sprites, Surfaces, and Rectangles

Das ist ein Sprite



Das Bild des Sprites wird  
durch ein pygame.Surface  
Objekt dargestellt



# Sprites, Surfaces, and Rectangles

Das ist ein Sprite



Das Bild des Sprites wird durch ein pygame.Surface Objekt dargestellt

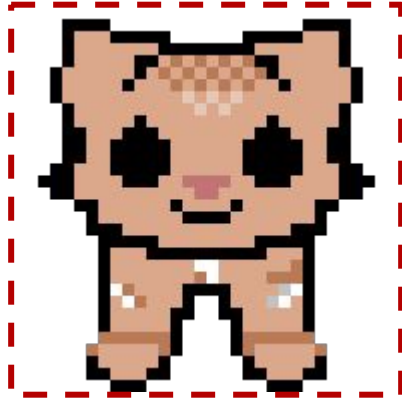
Das Sprite ist von einem imaginären Rechteck umgeben



Das imaginäre Rechteck wird durch ein pygame.Rect Objekt dargestellt

# Was du über Rectangles wissen musst

Das Sprite ist von einem imaginären Rechteck umgeben



Das imaginäre Rechteck wird durch ein `pygame.Rect` Objekt dargestellt

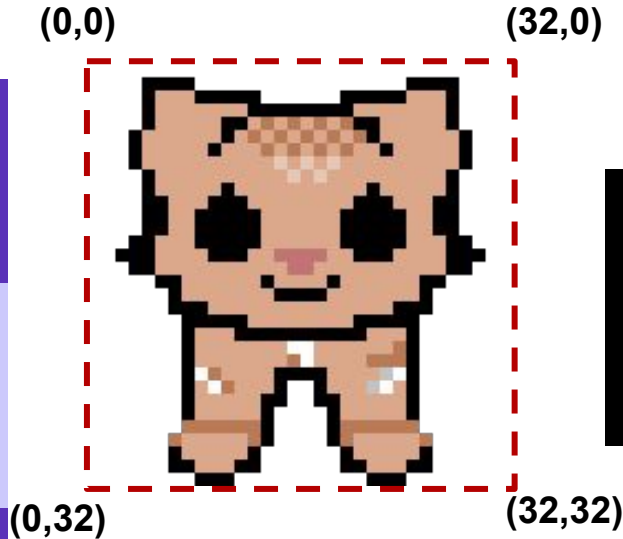
Wir nennen das imaginäre Rechteck des Sprites 'rect'.

Wie du dir vorstellen kannst, hat *rect* eine Breite und eine Höhe. Diese können wir leicht ermitteln

```
import pygame  
  
# upper left corner is at (0,0)  
rect = pygame.Rect(0, 0, 32, 32)  
  
print(rect.width, rect.height)
```

# Was du über Rectangles wissen musst

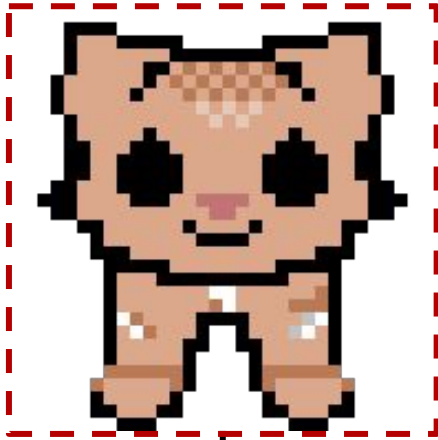
Der Zweck des imaginären Rechtecks des Sprites ist, dass wir die Koordinaten oder die Position des Sprites auf dem Bildschirm erfassen können.



```
print(rect.right) # x value of right side  
print(rect.left) # x value of left side  
print(rect.top) # y value of top side  
print(rect.bottom) # y value of bottom side
```

# Was du über Rectangles wissen musst

rect.centerx: 16



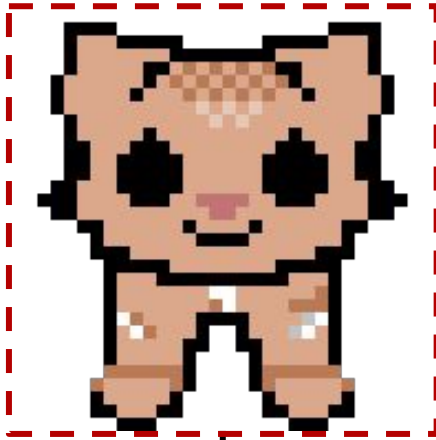
rect.midbottom: (16,32)

Der Zweck des imaginären Rechtecks des Sprites ist, dass wir die Koordinaten oder die Position des Sprites auf dem Bildschirm erfassen können. **(Verschiedene Positions-Attribute)**

```
x,y
top, left, bottom, right
topleft, bottomleft, topright,
bottomright
midtop, midleft, midbottom, midright
center, centerx, centery
size, width, height
w, h
```

# Was du über Rectangles wissen musst

rect.centerx: 50

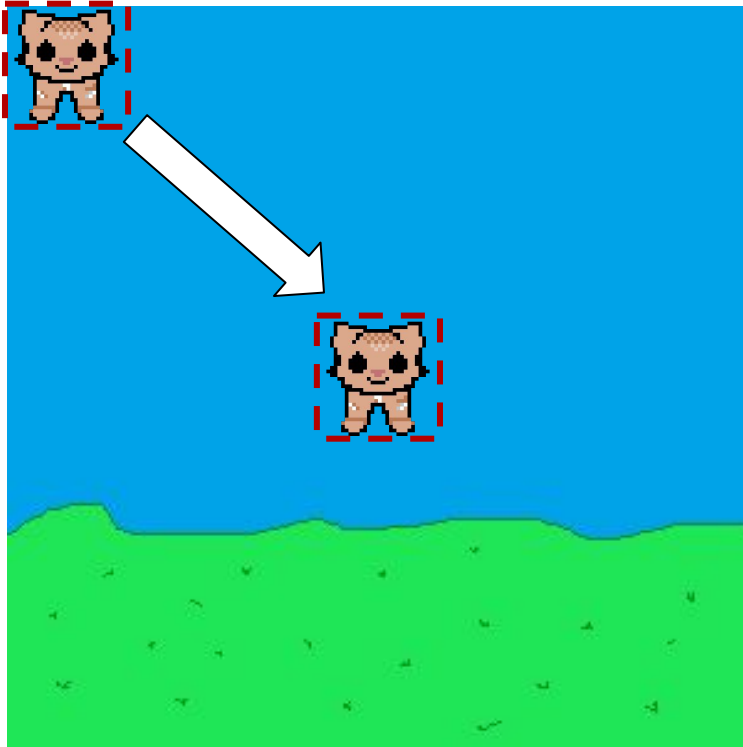


rect.midbottom:  
(50,100)

Das Sprite bewegt sich, wenn man einen Positions-Wert ändert

```
rect.midbottom = (50, 100)
print(rect.right) # x=66
print(rect.left) # x=34
print(rect.top) # y=68
print(rect.bottom) # y=100
```

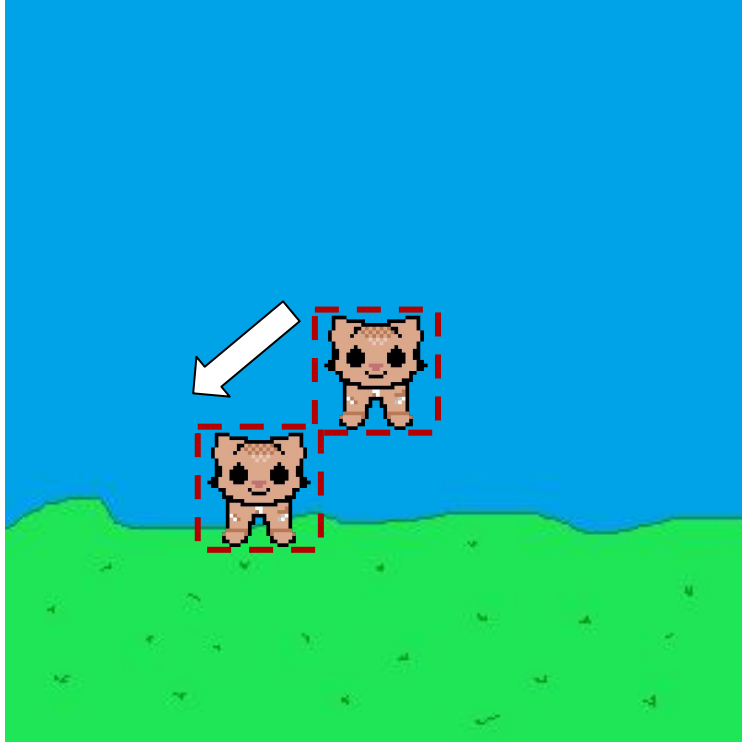
# Bewege den Sprite



```
rect.topleft = (0, 0)  
rect.center = (50, 50)
```

Dimensionen sind nicht exakt ;)

# Bewege den Sprite

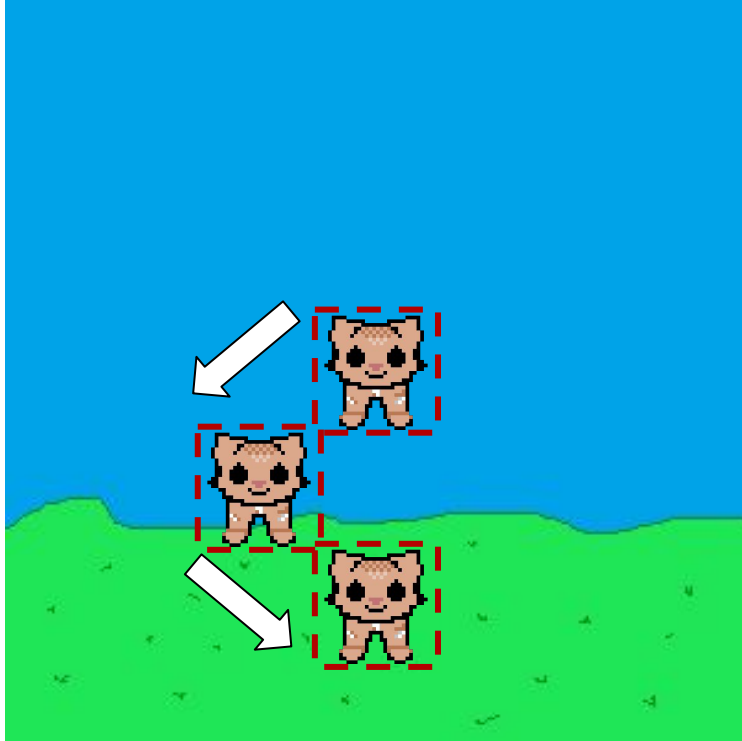


```
rect.center = (50, 50)
```

```
rect.topright = rect.bottomleft
```

Dimensionen sind nicht exakt ;)

# Bewege den Sprite



```
rect.center = (50, 50)
```

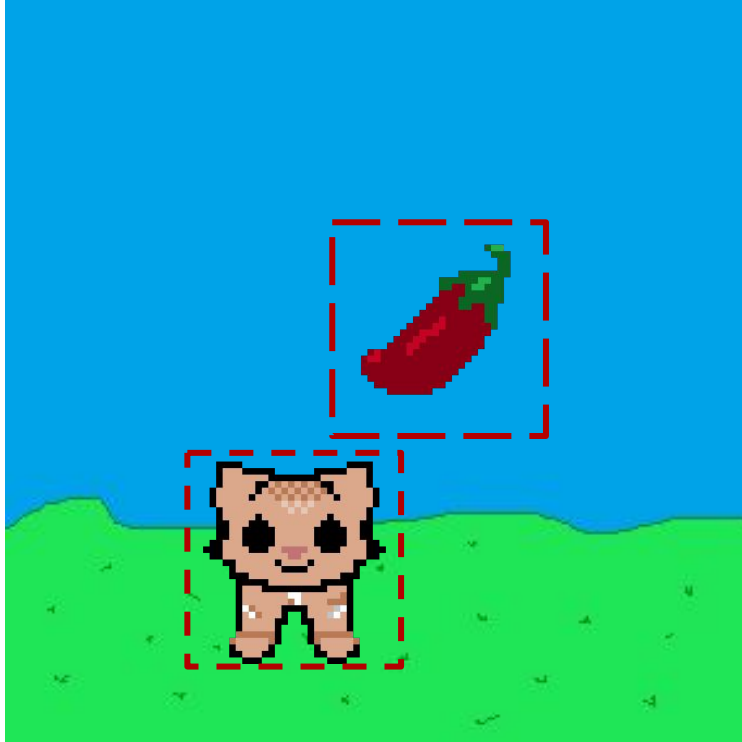
```
rect.topright = rect.bottomleft
```

```
rect.topleft = rect.bottomright
```

Dimensionen sind nicht exakt ;)



# Kollisionen



Wir haben zwei Sprites.

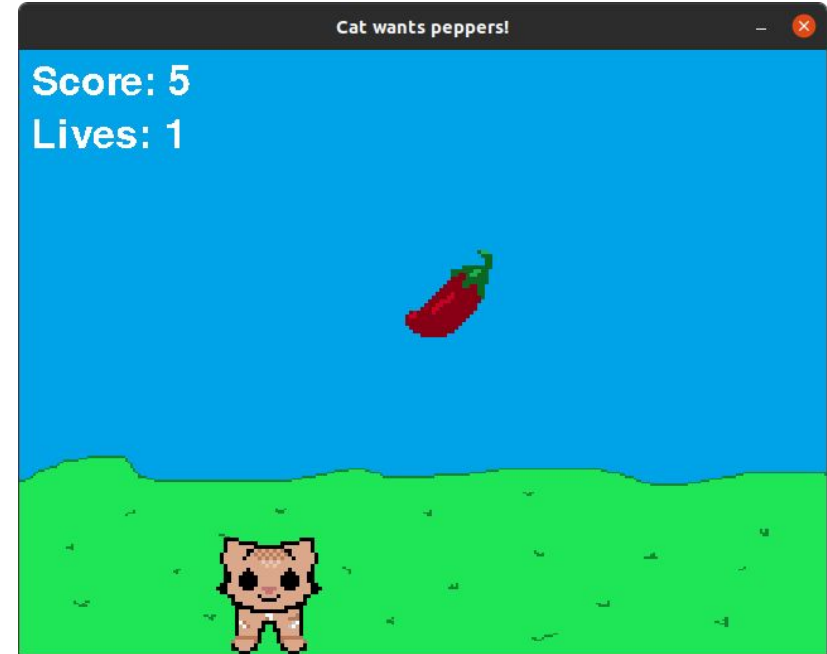
Das Rechteck der Cat heißt `cat_rect` und das Rechteck der Pepper heißt `pepper_rect`.

Wir können die Rechtecke der Sprites verwenden, um zu prüfen, ob sie sich berühren.

Wie können wir das tun?

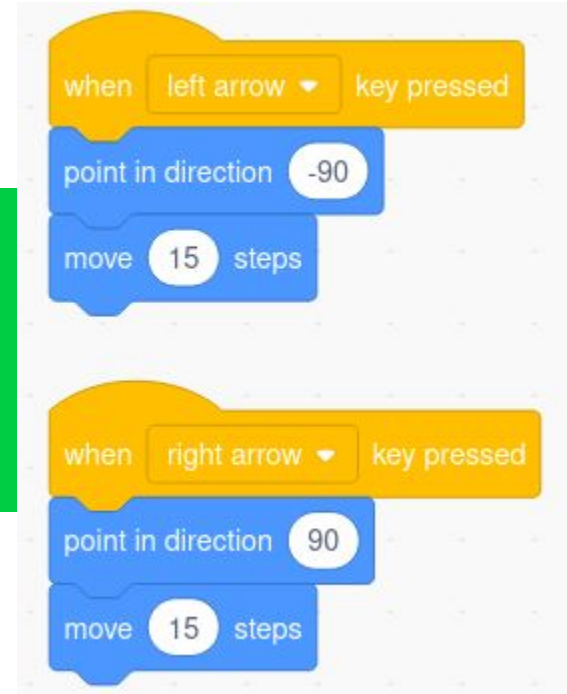
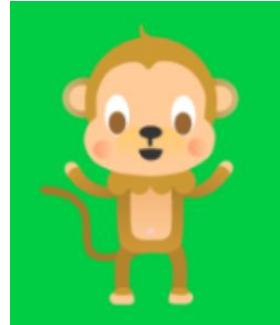
Versuche, dies auf einem Blatt Papier oder an der Tafel herauszufinden.

# Coding challenge



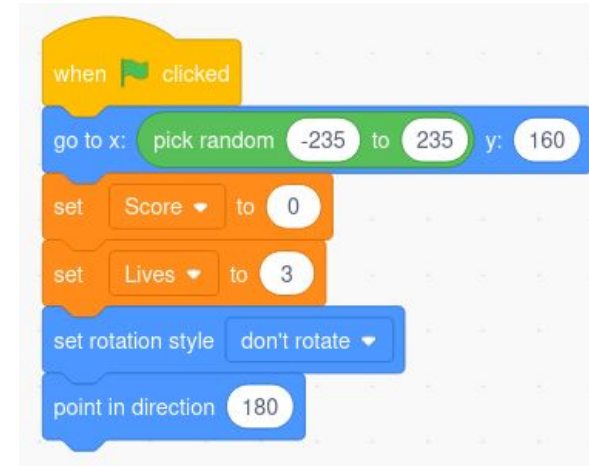
# Zuvor

- Der Affe kann sich von links nach rechts bewegen



# Zuvor

- Der Affe kann sich von links nach rechts bewegen
- Die Banane fällt nach unten (und nicht wie zuvor nach oben)



# Zuvor

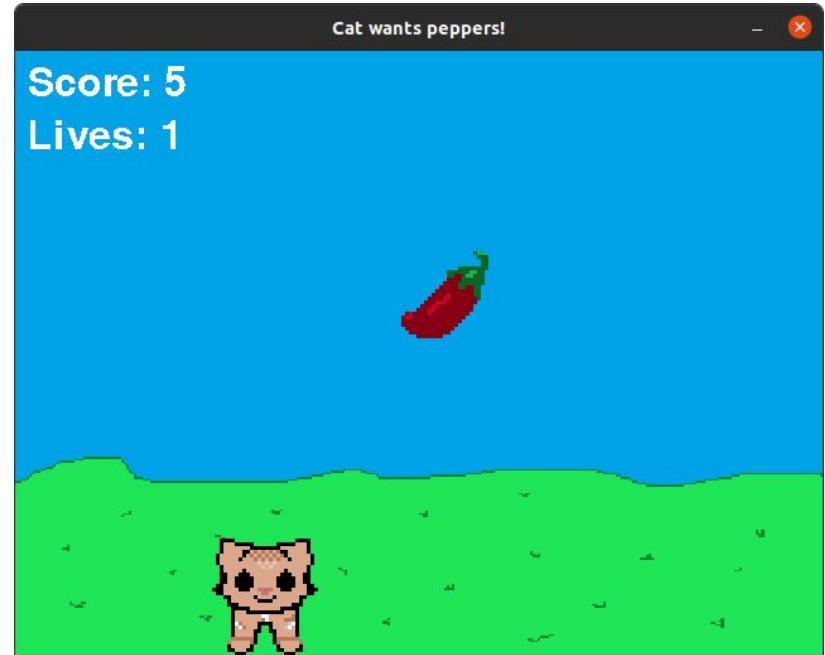
- Der Affe kann sich von links nach rechts bewegen
- Die Banane fällt nach unten (und nicht wie zuvor nach oben)
- Wir haben einen Fehler behoben, durch den die Bananen immer in die Mitte des Bildschirms wanderten, wenn der Spieler eine fängt.



# Weitere Fehler!

Jetzt haben wir etwas Python-Code und wir brauchen wieder eure Hilfe bei einigen Fehlern!

**Spiel:** Das Spiel ist dasselbe wie das Spiel "Affe und Bananen", nur mit einer Katze und einer Peperoni. Die Katze möchte die Peperoni fangen, weil sie weiß, dass ihre Besitzerin sie gerne isst.



# Getting started

Öffne den gesamten Ordner  
“Day\_2/Code/Lecture\_4\_coding\_challenge”  
in VS Code.

Um das Spiel zu starten, öffne ein Terminal  
im Ordner und gib **python Game.py** ein.

Wenn du das Spiel zum ersten Mal  
ausführst, wirst du feststellen, dass es nicht  
ganz richtig funktioniert...



# The Bugs

1. Die Pepper fliegt weg! Kannst du einen Fehler in dem Code finden, der das verursacht? Der Fehler ist irgendwo in **Pepper.py**.
2. Wir können den Spieler nicht bewegen! Wir brauchen deine Hilfe, damit sich der Spieler bewegen kann. Hilf uns, dies zu beheben, indem du die **move**-Funktion in **Player.py** vervollständigst.
3. Die Pepper geht immer in die Mitte des Bildschirms, wenn die Katze sie fängt! Hilf uns, den Code zu finden, der dies verursacht! Der Fehler liegt irgendwo in **Game.py**.

💡 Tipps: Die Fehlerhinweise stehen auch oben in jeder Datei und in der README-Datei. Löse die Bugs der Reihe nach, denn sie werden zunehmend schwieriger. Die Print-Funktion kann bei der Fehlersuche sehr hilfreich sein. Verwende sie als Hilfsmittel, um die Werte der Variablen auszugeben, wenn ein bestimmter Code ausgeführt wird, usw. Schaut euch den vorhandenen Code (ohne Fehler) an, um Ideen für eure Lösungen zu bekommen. **Wichtig ist, dass ihr euch gegenseitig hilft, Fragen stellt und Spaß habt!**



The Game Loop: Drawing and time-based updates

# Part 2

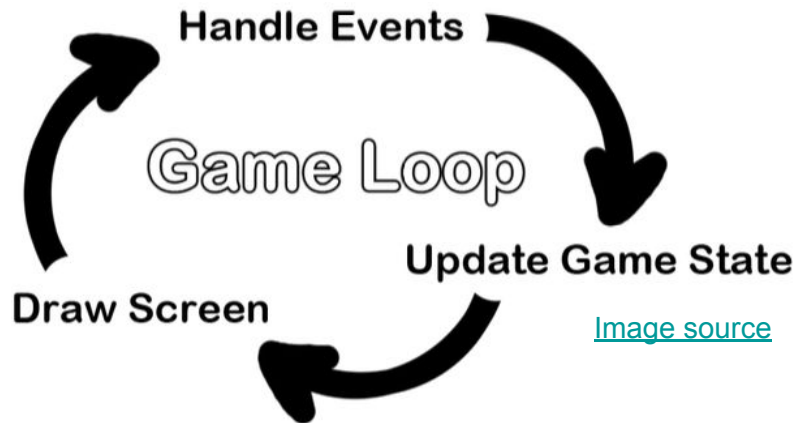
# Main game loop

Die “Game Loop” ist das Herz eines Videospiels.

Drei wichtige Dinge passieren in der Game Loop:

1. **Handle Events** (wie das Drücken einer Taste oder das Klicken der Maus)
2. **Update Game Status** (z. B. wo sich Pacman und die Geister befinden)
3. **Draw Screen** (macht das Spiel im Fenster sichtbar)

Eine Iteration der Game Loop ist ein *Frame*



**Mehr zu diesen drei Aufgaben und Schleifeniterationen jetzt!**

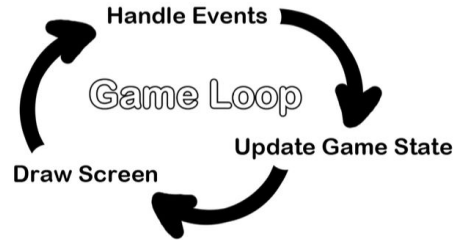
# Basic Game Loop in Pygame

```
import sys, pygame # import the pygame library
from pygame.locals import * # provides constant variables like QUIT

pygame.init() # needed for other pygame functions to work
GAME_SCREEN = pygame.display.set_mode((640, 480)) # dimensions of display
pygame.display.set_caption('Basic game loop') # Adds title to window

while True: # Game loop
    for event in pygame.event.get(): # get events and iterate through them
        if event.type == QUIT: # if user clicks exit button
            pygame.quit() # Deactivates the pygame library
            sys.exit() # ends the whole program
    pygame.display.update()
```

# Basic Game Loop in Pygame



**Handle Events:** Es wird nur 1 Ereignis bearbeitet (Beenden).

**Update Game State:** Wir haben kein Spiel, also gibt es auch keinen Spielstatus zu aktualisieren.

**Draw Screen:** Der Screen ist leer. Wenn wir wollen, dass etwas erscheint, müssen wir es auf die **GAME\_SCREEN**-Oberfläche zeichnen.

```
import sys, pygame
from pygame.locals import *

pygame.init()
GAME_SCREEN = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Basic game loop')

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```

# Drawing Primitives

Als erstes haben wir ein **pygame.Rect** hinzugefügt, das mit der **pygame.Surface GAME\_SCREEN** verbunden ist, so dass wir uns leicht auf Stellen auf dem Screen beziehen können.

Daher:

**screen\_rect.center** bezieht sich auf die (x,y) Position des zentralen Pixels auf dem gesamten Screen.

```
pygame.init()
screen_rect = pygame.Rect(0,0,640,480)
GAME_SCREEN = pygame.display.set_mode((screen_rect.width, screen_rect.height))
pygame.display.set_caption('Drawing')

WHITE = (255, 255, 255) # rgb color values.
pygame.draw.circle(GAME_SCREEN, WHITE, screen_rect.center, 20)

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```



# Drawing Primitives

Dann erstellen wir eine Variable **WHITE**, in der wir die rgb-Werte (rot, blau, grün) speichern, die im Computer die Farbe Weiß ergeben.

Betrachten wir die nächste Codezeile...

```
pygame.init()
screen_rect = pygame.Rect(0,0,640,480)
GAME_SCREEN = pygame.display.set_mode((screen_rect.width, screen_rect.height))
pygame.display.set_caption('Drawing')

WHITE = (255, 255, 255) # rgb color values.
pygame.draw.circle(GAME_SCREEN, WHITE, screen_rect.center, 20)

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
```



# Drawing Primitives

Was wir dem Computer sagen:

Zeichne einen Kreis auf der  
GAME\_SCREEN Oberfläche

Weiss  
machen

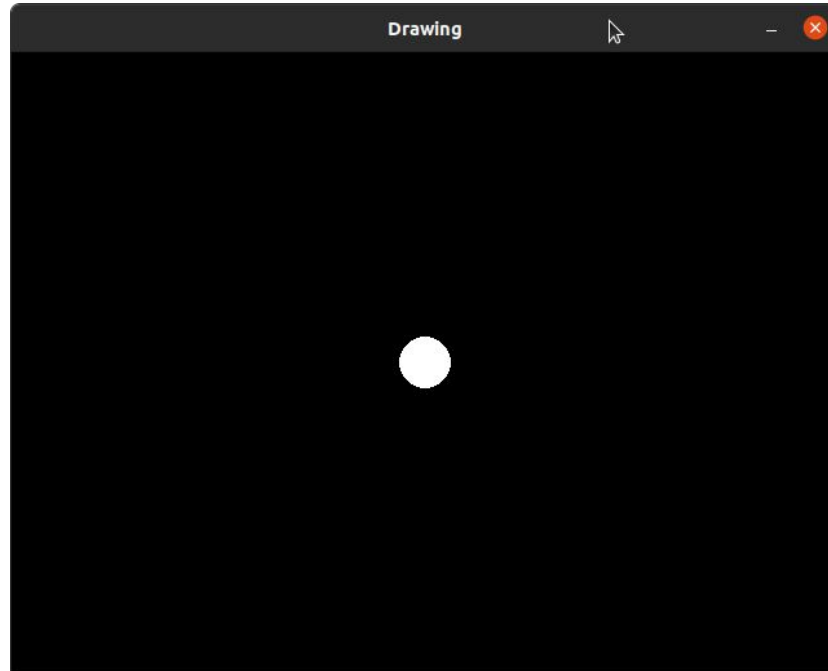
Der Radius  
ist 20 Pixel

```
pygame.draw.circle(GAME_SCREEN, WHITE, screen_rect.center, 20)
```

Die Mitte sollte sich  
in der Mitte des  
Bildschirms  
befinden.

# Einfache Objekte zeichnen: Ergebnis

```
pygame.draw.circle(GAME_SCREEN, WHITE, screen_rect.center, 20)
```





# Einfache Objekte zeichnen: Weitere Formen

```
WHITE = (255, 255, 255)
```

```
BLACK = (0, 0, 0)
```

```
RED = (255, 0, 0)
```

```
GREEN = (0, 255, 0)
```

```
BLUE = (0, 0, 255)
```

```
GAME_SCREEN.fill(WHITE)
```

```
pygame.draw.circle(GAME_SCREEN, BLACK, screen_rect.center, 20)
```

```
pygame.draw.rect(GAME_SCREEN, RED, (screen_rect.left + 40, screen_rect.top, 30, 50))
```

```
my_rect = pygame.Rect(screen_rect.centerx, # top left x  
                        screen_rect.centery + 20, # top left y  
                        screen_rect.width / 8, # width of my_rect  
                        screen_rect.height / 8) # height of my_rect  
pygame.draw.rect(GAME_SCREEN, BLUE, my_rect)
```

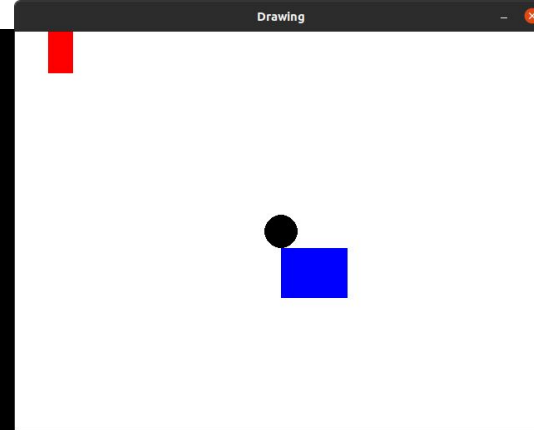
Viele weitere Formen können mit `pygame.draw` gezeichnet werden, wie z.B. **rect** (rectangle), **ellipse**, **arc**, **polygon**.

# Einfache Objekte zeichnen: Ergebnis

```
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)

GAME_SCREEN.fill(WHITE)

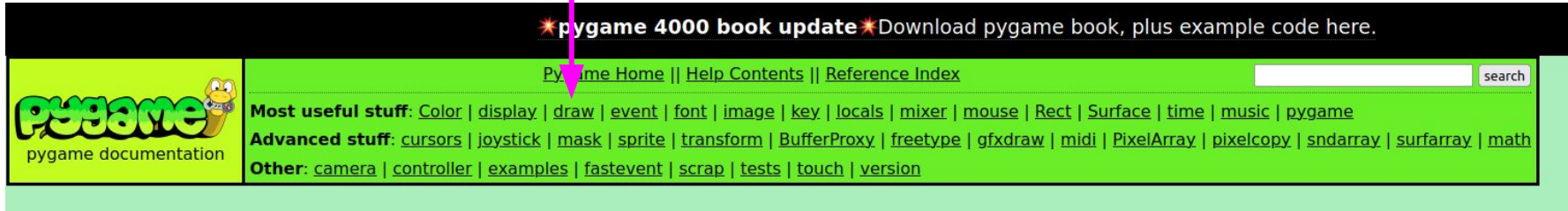
pygame.draw.circle(GAME_SCREEN, BLACK, screen_rect.center, 20)
pygame.draw.rect(GAME_SCREEN, RED, (screen_rect.left + 40, screen_rect.top, 30, 50))
my_rect = pygame.Rect(screen_rect.centerx, # top left x
                       screen_rect.centery + 20, # top left y
                       screen_rect.width / 8, # width of my_rect
                       screen_rect.height / 8) # height of my_rect
pygame.draw.rect(GAME_SCREEN, BLUE, my_rect)
```



# Pygame Dokumentation

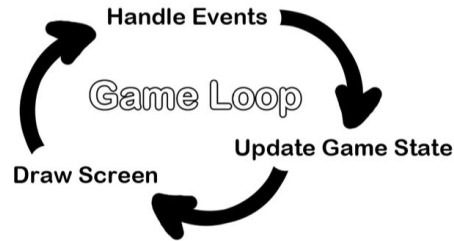
Für das Zeichnen anderer Formen, kannst du in der Pygame Dokumentation nachschauen

<https://www.pygame.org/docs/>



Es ist eine sehr hilfreiche Quelle für alles, was man über Pygame wissen muss ;)

# Zurück zur game loop.



Da Du nun ein wenig über das Zeichnen auf dem Bildschirm weißt, lass uns zu unserer Spielschleife zurückkehren.

Bevor wir beginnen, haben wir eine Uhr hinzugefügt, mit der wir steuern können, wie schnell die Spielschleife iteriert.

Eine Iteration ist ein Frame. Wenn wir also `clock.tick(1)` aufrufen, sagen wir, dass die Geschwindigkeit ein **Frame pro Sekunde (FPS)** sein soll

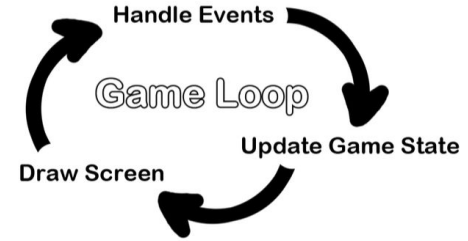
```
import sys, pygame
from pygame.locals import *

pygame.init()
clock = pygame.time.Clock()

GAME_SCREEN = pygame.display.set_mode((640, 480))
pygame.display.set_caption('Drawing in the loop')

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
    pygame.display.update()
    clock.tick(1)
```

# Update Game State & Draw Screen



Wir wollen ein rotes Rechteck vom oberen Rand des Bildschirms fallen lassen.

Was sind die Komponenten unserer Spielschleife?

**Handle events:** Betrachte den Ablauf der Zeit als ein Ereignis

**Update game state:** Die y-Position des Rechtecks soll sich erhöhen

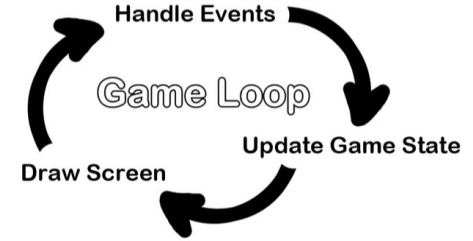
**Draw:** Zeichne das Rechteck an seiner richtigen Position

# Update Game State & Draw Screen

**Handle events:** Betrachte den Ablauf der Zeit als ein Ereignis

**Update game state:** Die y-Position des Rechtecks soll sich erhöhen

**Draw:** Zeichne das Rechteck an seiner richtigen Position



Stellen wir uns dieses rechteckige Objekt als Pepper vor :)

Vor der Game Loop müssen wir einige Eigenschaften der Pepper definieren.

**Wo befindet sich die Pepper nach Ausführung dieses Codes?**

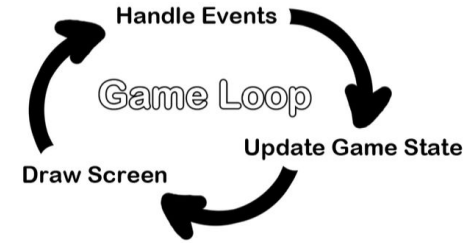
```
pepper_width = 80
pepper_height = 100
pepper_rect = pygame.Rect(0, 0,
                           pepper_width, pepper_height)
pepper_rect.centerx = screen_rect.centerx
pepper_rect.top = screen_rect.top
pepper_color = RED
pepper_speed = 20
```

# Update Game State & Draw Screen

**Handle events:** Betrachte den Ablauf der Zeit als ein Ereignis

**Update game state:** Die y-Position des Rechtecks soll sich erhöhen

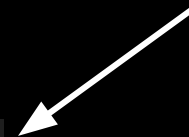
**Draw:** Zeichne das Rechteck an seiner richtigen Position



```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    """ Draw pepper """
    pygame.draw.rect(GAME_SCREEN, pepper_color, pepper_rect)
    pygame.display.update()
    clock.tick(1)
```

**Wir fügen den drawing part  
in die game loop direkt vor  
pygame.display.update() ein**

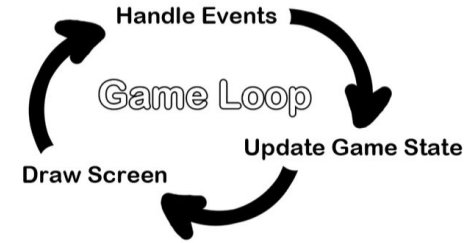


# Update Game State & Draw Screen

**Handle events:** Betrachte den Ablauf der Zeit als ein Ereignis

**Update game state:** Die y-Position des Rechtecks soll sich erhöhen

**Draw:** Zeichne das Rechteck an seiner richtigen Position



```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    """ Update pepper to make it fall. """
    pepper_rect.top = pepper_rect.top + pepper_speed

    """ Draw pepper """
    pygame.draw.rect(GAME_SCREEN, pepper_color, pepper_rect)
    pygame.display.update()
    clock.tick(1)
```

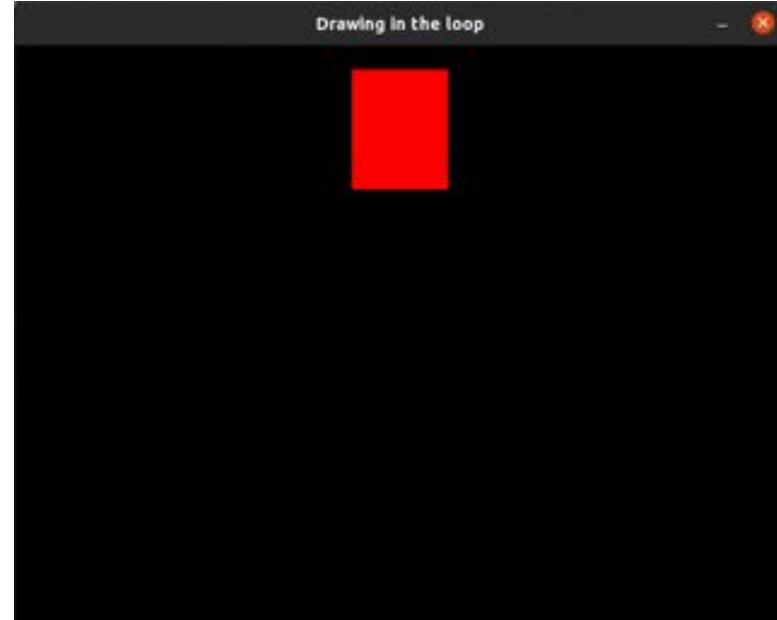
**Damit die Pepper fällt,  
müssen wir die Position  
updaten. Hinzufügen zum  
update code.**





# Was macht die Pepper?

Das scheint nicht richtig zu sein...



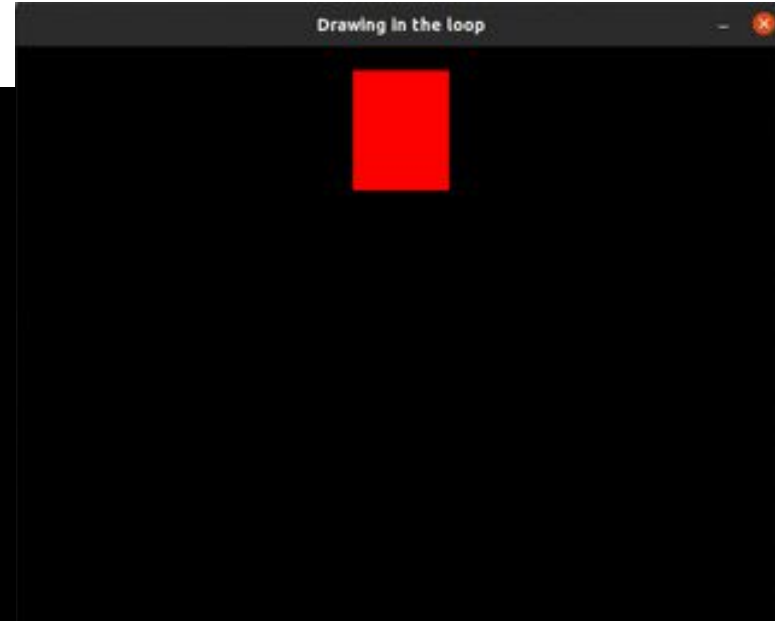
# Was macht die Pepper?

Unser Code zeichnet Rechtecke, aber die vorherige Zeichnung bleibt auf dem Screen!

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()


    """ Update pepper to make it fall. """
    pepper_rect.top = pepper_rect.top + pepper_speed

    """ Draw pepper """
    pygame.draw.rect(GAME_SCREEN, pepper_color, pepper_rect)
    pygame.display.update()
    clock.tick(1)
```



# Was können wir tun?

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        """ fill screen white """
        GAME_SCREEN.fill(WHITE)
        """ Update pepper to make it fall. """
        pepper_rect.top = pepper_rect.top + pepper_speed
        """ Draw pepper """
        pygame.draw.rect(GAME_SCREEN, pepper_color, pepper_rect)
        pygame.display.update()
        clock.tick(1)
```



Wie dies gehandhabt wird, hängt im Allgemeinen davon ab, welchen anderen Code wir haben und wie wir ihn implementiert haben.

Das gilt für Code im Allgemeinen, nicht nur für die Spieleprogrammierung ;)

Hier füllen wir in jeder Iteration den Screen mit weiß und verdecken damit alles, was vorher da war. Jetzt haben wir einen weißen Hintergrund!

# Was macht die Pepper?

Viel besser!

```
= 20

t in pygame.event.get():
    event.type == QUIT:
        pygame.quit()
        sys.exit()
    screen.white """
    EEN.fill(WHITE)
    te pepper to make it fall. """
    ect.top = pepper_rect.top + pepper_speed
    pepper """
    raw.rect(GAME_SCREEN, pepper_color, pepper_rect)
   isplay.update()
    ck(1)

DEBUG CONSOLE  TERMINAL  4: bash (L5)  + -
l2770:~/research/IT-summer-school-dev/lectures/L5$ python 1-Draw
```



# Verbesserung des Codes durch Klassen



```
pepper_width = 80
pepper_height = 100
pepper_rect = pygame.Rect(0, 0, pepper_width,
pepper_height)
pepper_rect.centerx = screen_rect.centerx
pepper_rect.top = screen_rect.top
pepper_color = RED
pepper_speed = 20
```



```
class Pepper:
    speed = 20
    color = RED

    def __init__(self, rect, midtop):
        self.rect = rect
        self.rect.midtop = midtop
```

```
pepper = Pepper(pygame.Rect(0, 0, 80, 100), screen_rect.midtop)
```

# Verbesserung des Codes durch Klassen



```
""" Update pepper to make it fall. """  
pepper_rect_top = pepper_rect_top + pepper_speed  
""" Draw pepper """  
pygame.draw.rect(GAME_SCREEN, pepper_color, pepper_rect)
```



```
""" Update pepper to make it fall. """  
pepper.rect.top = pepper.rect.top + pepper.speed  
""" Draw pepper """  
pygame.draw.rect(GAME_SCREEN, pepper.color, pepper.rect)
```

Jetzt greifen wir auf die Attribute des Pepper-Objekts zu, anstatt separate Variablen zu verwenden.

Wir können auf Pepper's Rect, Speed und Color über den Punkt zugreifen:

pepper.rect  
pepper.speed  
pepper.color

# update Funktion hinzufügen

```
class Pepper:
    speed = 20
    color = RED

    def __init__(self, rect, midtop):
        self.rect = rect
        self.rect.midtop = midtop

    def update(self):
        self.rect.top = self.rect.top +
self.speed
```

```
""" Update pepper to make it fall. """
pepper.rect.top = pepper.rect.top + pepper.speed
```

```
""" Update pepper to make it fall. """
pepper.update()
```



# draw Funktion hinzufügen

```
class Pepper:
    speed = 20
    color = RED

    def __init__(self, rect, midtop):
        self.rect = rect
        self.rect.midtop = midtop

    def update(self):
        self.rect.top = self.rect.top + self.speed

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

```
""" Update pepper """
pepper.update()

""" Draw pepper """
pepper.draw(GAME_SCREEN)
```



# Jetzt lassen sich einfach mehrere Peppers erzeugen :)

```
pepper = Pepper(pygame.Rect(0, 0, 80, 100), screen_rect.midtop)
pepper2 = Pepper(pygame.Rect(0, 0, 40, 50), (pepper.rect.left - 100, pepper.rect.top))
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    GAME_SCREEN.fill(WHITE)
    pepper.update()
    pepper2.update()
    pepper.draw(GAME_SCREEN)
    pepper2.draw(GAME_SCREEN)
    pygame.display.update()
    clock.tick(1)
```



# Pause?

The Game Loop: Condition-based updates

# Part 3

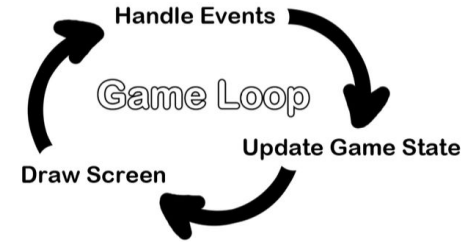
# Was bisher geschah

Wir haben eine ganze Game Loop erstellt um die Pepper herunterfallen zu lassen.

**Handle events:** Der Zeitablauf eines Ereignisses (und Beenden)

**Update game state:** Die y-Position des Rechtecks sollte sich erhöhen

**Draw:** Das Rectangle wird an der richtigen Position gezeichnet



```
me.time.Clock()

= pygame.Rect(0,0,640, 480)
= pygame.display.set_mode((screen_rect.width, screen_rect.height
ay.set_caption('Pepper Class Methods')

, 255, 255)
0, 0)

:
20
RED

it__(self, rect, midtop):
rect = rect
rect.midtop = midtop

DEBUG CONSOLE TERMINAL 4: bash (L5) + v
12770:~/research/IT-summer-school-dev/lectures/L5$ python 4-Two-P
```

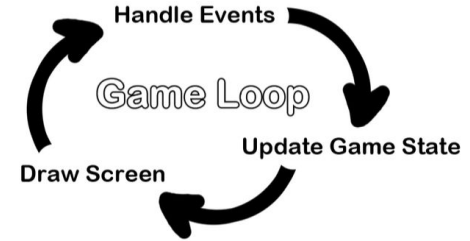
# Update game state: conditional

Wir möchten, dass die Peppers wieder an den oberen Rand des Screens zurückkehren, wenn sie den Boden berühren.

**Handle events:** Pepper berührt den Boden

**Update game state:** Die y-Position des Rectangles sollte zurückgesetzt werden

**Draw:** Das Rectangle wird an der richtigen Position gezeichnet



# Wie würde man dies umsetzen?

```
pepper.update()  
pepper2.update()  
  
""" Code for conditional update """  
  
pepper.draw(GAME_SCREEN)  
pepper2.draw(GAME_SCREEN)
```

Erhöhe die FPS auf 10, um dies schneller zu beobachten

# Wie würde man dies umsetzen? Lösung

```
pepper.update()
pepper2.update()

if pepper.rect.bottom >= screen_rect.bottom:
    pepper.rect.top = screen_rect.top
if pepper2.rect.bottom >= screen_rect.bottom:
    pepper2.rect.top = screen_rect.top

pepper.draw(GAME_SCREEN)
pepper2.draw(GAME_SCREEN)
```

Erhöhe die FPS auf 10, um dies schneller zu beobachten

# Wie würde man dies umsetzen? Lösung

```
pepper.update()
pepper2.update()

if pepper.rect.bottom >= screen_rect.bottom:
    pepper.rect.top = screen_rect.top
if pepper2.rect.bottom >= screen_rect.bottom:
    pepper2.rect.top = screen_rect.top

pepper.draw(GAME_SCREEN)
pepper2.draw(GAME_SCREEN)
```

**Wiederholter Code!**  
**Was können wir da machen?**



# Wiederholter Code! Was können wir da machen? Listen!

```
pepper = Pepper(pygame.Rect(0, 0, 80, 100), screen_rect.midtop)
pepper2 = Pepper(pygame.Rect(0, 0, 40, 50),
                  (pepper.rect.left - 100, pepper.rect.top))
pepper_list = [pepper, pepper2]
```

```
for item in pepper_list:
    item.update()

    if item.rect.bottom >= screen_rect.bottom:
        item.rect.top = screen_rect.top

    item.draw(GAME_SCREEN)
```

```
class PepperGroup:
    def __init__(self):
        self.items = []

    def add(self, item):
        self.items.append(item)

    def update(self):
        for item in self.items:
            item.update()

    def draw(self, surface):
        for item in self.items:
            item.draw(surface)

    def touch_ground_update(self, ground_rect):
        for item in self.items:
            if item.rect.bottom >= ground_rect.bottom:
                item.rect.top = ground_rect.top
```

# Noch besser...

Code: 11-Group-Class.py



# Wiederholter Code! Was können wir da machen?

```
pepper_list = PepperGroup()

pepper = Pepper(pygame.Rect(0, 0, 80, 100), screen_rect.midtop)
pepper2 = Pepper(pygame.Rect(0, 0, 40, 50),
                  (pepper.rect.left - 100, pepper.rect.top))

pepper_list.add(pepper)
pepper_list.add(pepper2)
```

Jetzt haben wir eine Sammlung von Peppers!

# Unser game loop 🥰

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    GAME_SCREEN.fill(WHITE)

    pepper_list.update()
    pepper_list.touch_ground_update(screen_rect)
    pepper_list.draw(GAME_SCREEN)

    pygame.display.update()
    clock.tick(10)
```

Jetzt ist unser game loop so einfach und schön 🥰

The Game Loop: Updates based on player events

# Part 4

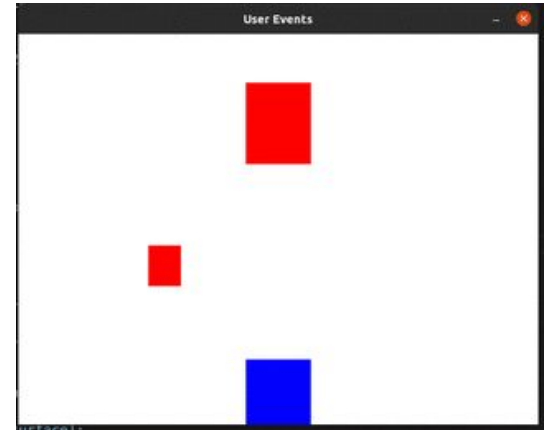
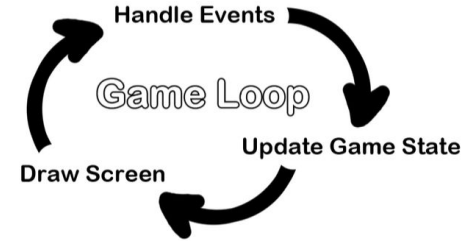
# Update game state: Player events

Wir möchten einen Player hinzufügen, den wir mit der Tastatur steuern können. Wenn wir die linke und rechte Pfeiltaste (oder die Tasten a und d) drücken, bewegt sich der Player nach links und rechts.

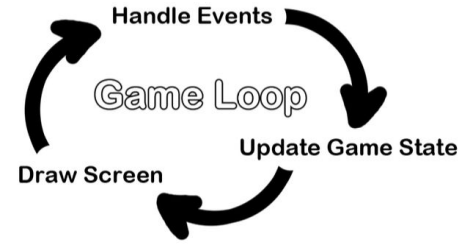
**Handle events:** Der Spieler drückt die Tasten links, rechts, 'a' oder 'd'.

**Update game state:** Die x-Position des Spielers wird aktualisiert, um ihn in die gewünschte Richtung zu bewegen

**Draw:** Zeichne den Spieler in seiner richtigen Position



# Update game state: Player events



**Handle events:** Der Spieler drückt die Tasten links, rechts, 'a' oder 'd'.

**Update game state:** Die x-Position des Spielers wird aktualisiert, um ihn in die gewünschte Richtung zu bewegen

**Draw:** Zeichne den Spieler in seiner richtigen Position

Wir fangen an, indem wir einen Spieler in den Code einfügen!

Lasst uns eine Player-Klasse erstellen :)

Sie ist der Pepper-Klasse sehr ähnlich, also fangen wir dort an und nehmen einige Änderungen vor!

# Player Class

```
class Player:
    speed = 10
    color = BLUE

    def __init__(self, rect, midbottom):
        self.rect = rect
        self.rect.midbottom = midbottom

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

Wir setzen die Geschwindigkeit des Spielers auf 10 und die Farbe auf blau.

Wir wollen den Spieler am Boden haben, also legen wir die Position auf der Basis von midbottom fest.

Die Methode touch\_ground\_update aus der Pepper-Klasse brauchen wir nicht.

Der Player fällt nicht, also lassen wir die Update-Methode erst einmal leer.



# Add a player to the loop

```
player = Player(pygame.Rect(0, 0, 80, 80), screen_rect.midbottom)
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        GAME_SCREEN.fill(WHITE)

    pepper_list.update()
    pepper_list.touch_ground_update(screen_rect)
    pepper_list.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)
    pygame.display.update()
    clock.tick(10)
```

Zuerst müssen wir ein Objekt der Klasse Player erstellen, das wir "player" nennen

Dann können wir die Funktion Player.draw in der Game Loop aufrufen, damit sie auf dem Bildschirm erscheint.

# Player Class

```
class Player:
    """ Player class continued """

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def move(self):
        return
```

Der Player braucht eine move Funktion.

# Player Class

```
class Player:
    """ Player class continued """

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def move(self, keystates):
        return
```

Der Player braucht eine **move** Funktion.

Um zu wissen, wie und in welche Richtung wir uns bewegen sollen, müssen wir wissen, welche Tasten gedrückt werden.

Fügen wir einen Parameter namens **keystates** hinzu, um Informationen über den Zustand der Tasten zu übergeben. Die Tastenzustände sind entweder "gedrückt" oder "nicht gedrückt". Das heißt, unsere Informationen liegen in Form von Booleschen Werten vor!

# Add a player to the loop

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        GAME_SCREEN.fill(WHITE)
        pepper_list.update()

        keystates = pygame.key.get_pressed()
        player.move(keystates)

        pepper_list.touch_ground_update(screen_rect)
        pepper_list.draw(GAME_SCREEN)
        player.draw(GAME_SCREEN)

    pygame.display.update()
    clock.tick(10)
```

In der Game Loop verwenden wir eine hilfreiche Funktion aus der Pygame-Bibliothek, die uns sagt, ob die Tasten gedrückt sind oder nicht! Sie gibt eine Liste von [False, True, False, ..., False] zurück, wobei jeder Index einer bestimmten Taste entspricht.

Dann können wir die Funktion `Player.move()` in der Game Loop aufrufen und ihr die von `pygame.key.get_pressed()` zurückgegebenen Tastenzustände übergeben

# Add a player to the loop

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        GAME_SCREEN.fill(WHITE)
        pepper_list.update()

    keystates = pygame.key.get_pressed()
    player.move(keystates)

    pepper_list.touch_ground_update(screen_rect)
    pepper_list.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)

    pygame.display.update()
    clock.tick(10)
```

**Nach dem Hinzufügen dieser beiden Zeilen  
wird nichts mehr passieren, da wir noch die  
Methode `Player.move` implementieren müssen**

# Player Class

```
class Player:
    """ Player class continued """

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def move(self, keystates):
        left_arrow_pressed = keystates[K_LEFT]
        right_arrow_pressed = keystates[K_RIGHT]
```

Glücklicherweise müssen wir die Indizes der Tastaturtasten in der Liste der **keystates** nicht erraten...

# Player Class

```
# this imports all pygame.locals
from pygame.locals import *
# Alternatively, we can specify which ones we want
from pygame.locals import K_LEFT, K_RIGHT, QUIT

class Player:
    """ Player class continued """

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)

    def move(self, keystates):
        left_arrow_pressed = keystates[K_LEFT]
        right_arrow_pressed = keystates[K_RIGHT]
```

Glücklicherweise müssen wir die Indizes der Tastaturtasten in der Liste der **keystates** nicht erraten...

... weil diese in konstanten Variablen gespeichert werden, die wir aus pygame.locals importieren (oben in der Datei)

# Pygame Dokumentation

Die Variablennamen, die die Indizes aller Schlüssel speichern, findest du in der pygame Dokumentation:

<https://www.pygame.org/docs/ref/key.html>

Das sieht so aus →

pygame Constant	ASCII	Description
K_BACKSPACE	\b	backspace
K_TAB	\t	tab
K_CLEAR		clear
K_RETURN	\r	return
K_PAUSE		pause
K_ESCAPE	^[	escape
K_SPACE		space
K_EXCLAIM	!	exclaim
K_QUOTEDBL	"	quotedbl
K_HASH	#	hash
K_DOLLAR	\$	dollar
K_AMPERSAND	&	ampersand
K_QUOTE	'	quote
K_LEFTPAREN	(	left parenthesis
K_RIGHTPAREN	)	right parenthesis
K_ASTERISK	*	asterisk
K_PLUS	+	plus sign
K_COMMA	,	comma
K_MINUS	-	minus sign
K_PERIOD	.	period
K_SLASH	/	forward slash
K_0	0	0
K_1	1	1
K_2	2	2
K_3	3	3
K_4	4	4
K_5	5	5
K_6	6	6
K_7	7	7
K_8	8	8
K_9	9	9
K_COLON	:	colon
K_SEMICOLON	;	semicolon
K_LESS	<	less-than sign
K_EQUALS	=	equals sign
K_GREATER	>	greater-than sign
K_QUESTION	?	question mark
K_AT	@	at
K_LEFTBRACKET	[	left bracket
K_BACKSLASH	\	backslash
K_RIGHTBRACKET	]	right bracket
K_CARET	^	caret
K_UNDERSCORE	_	underscore
K_BACKQUOTE	`	grave
K_a	a	a
K_b	b	b
K_c	c	c
K_d	d	d
K_e	e	e
K_f	f	f
K_g	g	g
K_h	h	h
K_i	i	i



# Player Class

```
class Player:
    """ Player class continued """
    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            print("move left")
        if keystates[K_RIGHT] or keystates[K_d]:
            print("move right")
```

Wir erstellen ein Template für die benötigten bedingten Anweisungen

**Links:** Wir wollen uns entweder mit der linken Pfeiltaste (für Rechtshänder) oder mit der Taste "a" (für Linkshänder) nach links bewegen können.

**Rechts:** Wir wollen uns entweder mit dem Pfeil nach rechts oder mit der Taste 'd' nach rechts bewegen können.

Versuche den Code jetzt auszuführen und vergewissere dich, dass "move left" im Terminal ausgegeben wird, wenn du den linken Pfeil oder die 'a'-Taste drückst, und "move right", wenn du den rechten Pfeil oder die 'd'-Taste drückst

# Tipp

Setze die Farbe der Pepper auf weiß, damit du dich auf die Bewegung im Spiel konzentrieren kannst.

```
class Pepper:  
    speed = 20  
    # color = RED  
    color = WHITE
```

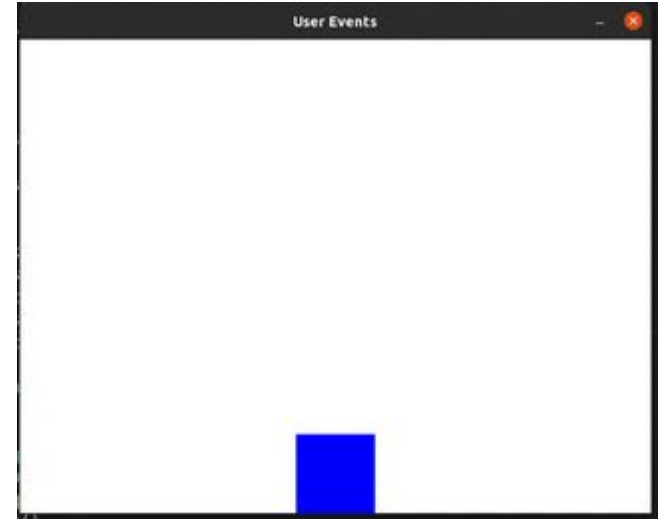
# Player Class: Beende die move-Funktion

Implementiert nun den Positionswechsel. Ihr wisst bestimmt bereits, wie das funktioniert. Schau dir dazu an, wie sich die Position der Pepper ändert und wie dies für den Player angepasst werden muss.

```
class Player:
    """ Player class continued """
    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            """ Fill in code to move left """
        if keystates[K_RIGHT] or keystates[K_d]:
            """ Fill in code to move right """
```

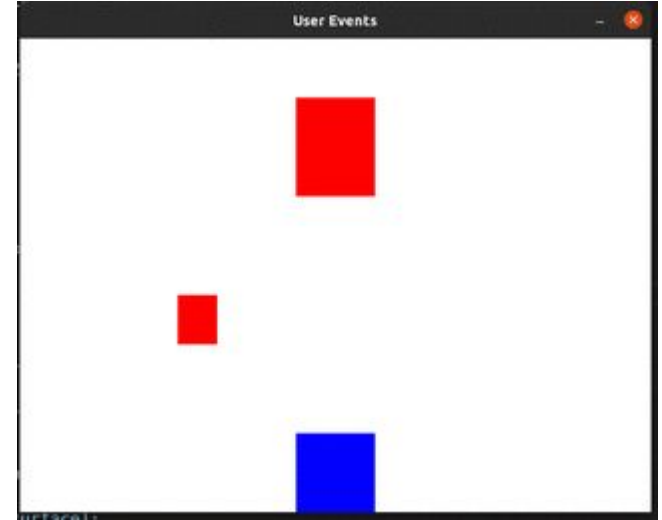
# Player Class: Lösung

```
class Player:
    """ Player class continued """
    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
            self.rect.x = self.rect.x + self.speed
```



# Player Class: Füge die Peppers wieder ein

```
class Pepper:  
    speed = 20  
    color = RED  
    # color = WHITE
```

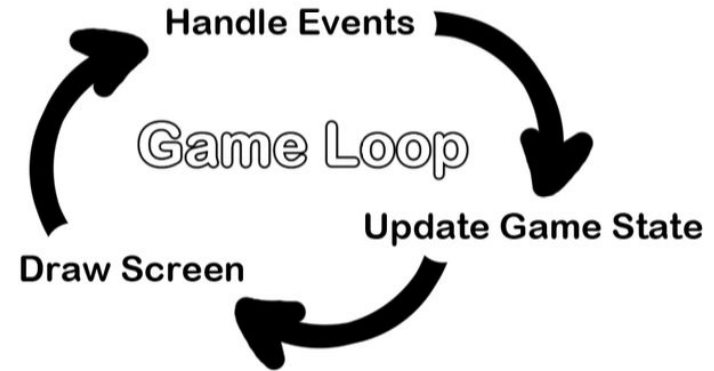


# Kurze Wiederholung: Game Loop

Die “Game Loop” ist das Herz eines Videospiele

Die Game Loop macht 3 Dinge

1. **Handles events**
2. **Updates the state of the game**
3. **Draws the Screen**



# Review: Update the game state

## Event type: Die Zeit vergeht

Wir steuern die Zeit, die vergeht, mit `clock.tick()`, so dass die Zeit vergeht, wenn die Spielschleife eine Iteration abschließt.

## Update: ?

Was updaten wir und wie?

# Review: Update the game state

## Event type: Die Zeit vergeht

Wir steuern die Zeit, die vergeht, mit `clock.tick()`, so dass die Zeit vergeht, wenn die Spielschleife eine Iteration abschließt.

## Update: Pepper state

Um das Fallen der Pepper zu simulieren, erstellen wir eine `update` Funktion für die Pepper-Klasse, die die y-Koordinatenposition der Pepper auf der Grundlage der Geschwindigkeit der Pepper (in Pixel pro Frame) erhöht.



# Review: Update the game state

**Event type: Spielbedingung erfüllt      Update: ?**

Die Pepper berührt den Boden.  
Wir überprüfen dies mit einer bedingten Anweisung, um herauszufinden, ob die y-Koordinate der Unterseite der Peppers größer oder gleich der Unterseite des Screens ist.

# Review: Update the game state

## Event type: Game condition met

Die Pepper berührt den Boden. Wir überprüfen dies mit einer bedingten Anweisung, um herauszufinden, ob die y-Koordinate der Unterseite der Peppers größer oder gleich der Unterseite des Screens ist.

## Update: Pepper state

Die Pepper wird am oberen Rand des Bildschirms neu generiert. Um dies zu implementieren, setzen wir die `Pepper.rect.midtop` (x,y)-Koordinaten gleich den mittleren (x,y)-Koordinaten des Screens.

# Review: Update the game state

**Event type: User click**

**Update:**

Der user klickt den **QUIT**  button  
im Spielfenster.



# Review: Update the game state

## Event type: User click

Der user klickt den **QUIT**  button im Spielfenster.

## Update: Quit game

Wir deaktivieren Pygame mit `pygame.quit()` und beenden dann das Python-Programm mit `sys.exit()`



# Review: Update the game state

**Event type: User drückt  
Tasten**

**Update:**

Der User drückt die Pfeile nach links oder rechts oder die Tasten 'a' oder 'd'.



# Review: Update the game state

## Event type: User drückt Tasten

Der User drückt die Pfeile nach links oder rechts oder die Tasten 'a' oder 'd'.

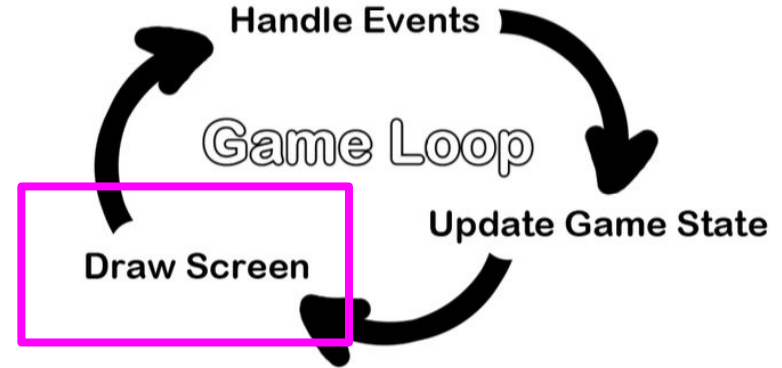
## Update: Player state

Die Position des Spielers wird geändert, indem sie entweder nach links oder nach rechts verschoben wird. Dazu wird die `Player.rect.x`-Position um die `Player.speed` (Pixel pro Frame) verringert oder erhöht.



# Review: Last but not least? Draw Screen!

1. Im Zeitverlauf
  - a. Die Positionen der Paprika werden immer mit der Zeit aktualisiert
2. Bedingungen des game state
  - a. Wenn die Pepper den Boden berührt
3. User events
  - a. Klicke QUIT button
  - b. Drücke Tasten

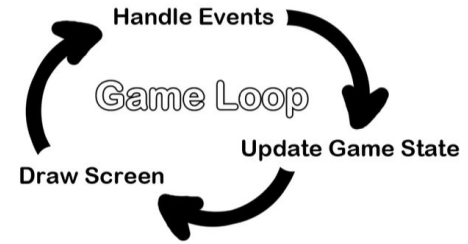


# Abschließende Gedanken...





# Die Logik der Spieleprogrammierung



Der Schlüssel zum **Verständnis** der Logik der Spieleprogrammierung ist das Verständnis der **Game Loop**.

Wenn ihr euer eigenes Spiel entwickeln wollt, könnt ihr die Logik eurer Programmierung planen, indem ihr an die 3 Teile der Spielschleife denkt.

**Frage dich:** Was soll im Spiel passieren?

**Dann überlege:**

- Welches **Ereignis** soll dies auslösen?
- Welche Variablen und Spielobjekte müssen **aktualisiert** werden und wie?
- Was wird auf dem Screen **angezeigt**?

# Code kann verschieden aussehen!

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom):
        super().__init__()

        self.image = image
        self.rect =
self.image.get_rect(midbottom=midbottom)
        self.speed = 10

    def move(self, keystate):
        left_arrow_pressed = keystate[K_LEFT]
        right_arrow_pressed = keystate[K_RIGHT]

        if left_arrow_pressed:
            self.rect.x = self.rect.x - self.speed

        if right_arrow_pressed:
            self.rect.x += self.speed
```

```
class Player:
    speed = 10
    color = BLUE

    def __init__(self, rect, midbottom):
        self.rect = rect
        self.rect.midbottom = midbottom

    def draw(self, surface):
        pygame.draw.rect(surface, self.color,
self.rect)

    def move(self, keystates):
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
            self.rect.x = self.rect.x + self.speed
```

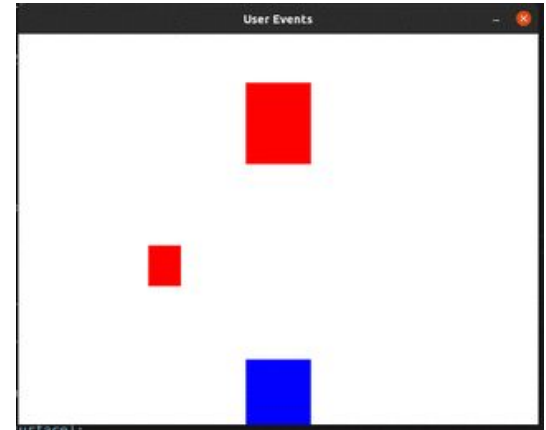
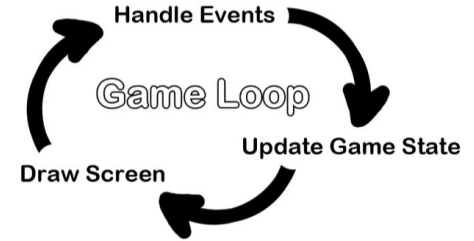
# Extra time?

Es fehlt eine Bedingung! Die Peppers müssen auch oben wieder auftauchen, wenn sie den Spieler berühren. Kannst du das lösen?

**Handle events:**

**Update game state:**

**Draw:**



Class inheritance

# Part 5

# Erstelle eine Sprite base class

- Was haben der Player und die Pepper-Klasse gemeinsam?
  - Rect
  - Color
  - Update
  - Draw
- Wir können den Code noch weiter vereinfachen, indem wir eine übergeordnete Klasse erstellen, von der Player und Pepper erben.




# Step 1: Gemeinsamkeiten in die Sprite-Klasse verschieben

```
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```



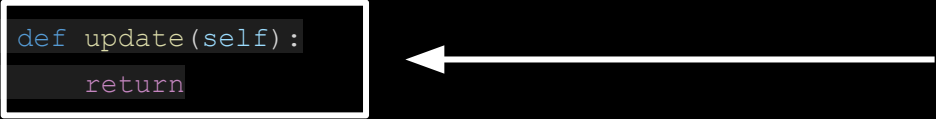
**Konstruktor:** Sowohl Pepper als auch Player haben ein Rechteck und eine Farbe. Die Rechtecke und Farben können unterschiedlich sein, also machen wir sie zu Parametern, über die wir entscheiden können, wenn wir unsere Objekte erstellen.

# Step 1: Gemeinsamkeiten in die Sprite-Klasse verschieben

```
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```



**Konstruktor:** Übergibt das gewünschte rect und die color, wenn wir ein Pepper- oder Player-Objekt erstellen.

**Update-Funktion:** Unsere Sprites werden auf unterschiedliche Weise aktualisiert, daher können wir diese Methode unimplementiert lassen und sie in den Pepper- und Player-Klassen außer Kraft setzen

# Step 1: Gemeinsamkeiten in die Sprite-Klasse verschieben

```
class Sprite:
    def __init__(self, rect, color):
        self.rect = rect
        self.color = color

    def update(self):
        return

    def draw(self, surface):
        pygame.draw.rect(surface, self.color, self.rect)
```

**Konstruktor:** Übergibt das gewünschte rect und die color, wenn wir ein Pepper- oder Player-Objekt erstellen.

**Update-Methode:** Wir werden diese Methode individuell für die Kind-Klassen implementieren.

**Draw-Methode:** In unserem Spiel werden die Sprites auf die gleiche Weise gezeichnet. Daher können wir die Implementierung in die Basisklasse aufnehmen und sie aus Pepper und Player entfernen.



## Step 2: Untergeordnete Klassen reparieren

**Vererbung:** Wir definieren die Pepper-Klasse wie folgt. Dann erbt Pepper alle Attribute und Methoden von der Sprite-Klasse.

```
class Pepper(Sprite):  
    def __init__(self, rect, color=RED, speed=10):  
        super().__init__(rect, color)  
        self.speed = speed  
  
    def update(self):  
        self.rect.top = self.rect.top + self.speed  
  
    def touch_ground_update(self, ground_rect):  
        if self.rect.bottom >= ground_rect.bottom:  
            self.rect.top = ground_rect.top
```

# Step 2: Untergeordnete Klassen reparieren

```
class Pepper(Sprite):  
    def __init__(self, rect, color=RED, speed=10):  
        super().__init__(rect, color)  
        self.speed = speed  
  
    def update(self):  
        self.rect.top = self.rect.top + self.speed  
  
    def touch_ground_update(self, ground_rect):  
        if self.rect.bottom >= ground_rect.bottom:  
            self.rect.top = ground_rect.top
```

**Vererbung:** Pepper erbt von der Klasse Sprite.

**Konstruktor:** Wir können `super().__init__()` aufrufen, der den Konstruktor der übergeordneten Sprite-Klasse verwendet. Wir geben die gewünschten Werte für `rect` und `color` ein. Wir fügen auch einen Geschwindigkeitsparameter hinzu. Wir geben Standardwerte für die Farbe und die Geschwindigkeit an, so dass wir sie ändern können, wenn wir es wollen. Aber im Allgemeinen werden unsere Pepper rot sein und mit der gleichen Geschwindigkeit fallen.

# Step 2: Untergeordnete Klassen reparieren

```
class Pepper(Sprite):  
    def __init__(self, rect, color=RED, speed=10):  
        super().__init__(rect, color)  
        self.speed = speed  
  
    def update(self):  
        self.rect.top = self.rect.top + self.speed  
  
    def touch_ground_update(self, ground_rect):  
        if self.rect.bottom >= ground_rect.bottom:  
            self.rect.top = ground_rect.top
```

**Vererbung:** Pepper erbt von der Klasse Sprite.

**Konstruktor:** Wir können `super().__init__()` aufrufen, der den Konstruktor der übergeordneten Sprite-Klasse verwendet. Füge zusätzliche Parameter der Kind-Klasse hinzu

**Update method:** Bleibt gleich.

# Step 2: Untergeordnete Klassen reparieren

```
class Pepper(Sprite):  
    def __init__(self, rect, color=RED, speed=10):  
        super().__init__(rect, color)  
        self.speed = speed  
  
    def update(self):  
        self.rect.top = self.rect.top + self.speed  
  
    def touch_ground_update(self, ground_rect):  
        if self.rect.bottom >= ground_rect.bottom:  
            self.rect.top = ground_rect.top
```

**Vererbung:** Pepper erbt von der Klasse Sprite.

**Konstruktor:** Wir können `super().__init__()` aufrufen, der den Konstruktor der übergeordneten Sprite-Klasse verwendet. Füge zusätzliche Parameter der Kind-Klasse hinzu

**Update method:** Bleibt gleich.

**Zusätzliche Methoden:** Das Fallverhalten ist einzigartig für die Pepper-Klasse, daher belassen wir es in der Klasse.

# Step 2: Untergeordnete Klassen reparieren

```
class Player(Sprite):  
    def __init__(self, rect, color=BLUE, speed=10):  
        super().__init__(rect, color)  
        self.speed = speed  
  
    def move(self, keystates):  
        if keystates[K_LEFT] or keystates[K_a]:  
            self.rect.x = self.rect.x - self.speed  
        if keystates[K_RIGHT] or keystates[K_d]:  
            self.rect.x = self.rect.x + self.speed
```

**Mache das gleiche mit der Player Klasse...**

**Vererbung:** Player erbt von der Klasse Sprite.

**Konstruktor:** Wir können `super().__init__()` aufrufen, der den Konstruktor der übergeordneten Sprite-Klasse verwendet. Füge zusätzliche Parameter der Kind-Klasse hinzu

**Update method:** Wir brauchen keine spezielle Update Method für Player.

**Zusätzliche Methoden:** Die Bewegung des Players beim Drücken von Tasten ist einzigartig für den Player.

# Observe

```
class Sprite:
    def __init__(self, rect, color):
        print("Sprite.__init__")
        self.rect = rect
        self.color = color

    def update(self):
        print("Sprite.update")
        return

    def draw(self, surface):
        print("Sprite.draw")
        pygame.draw.rect(surface, self.color, self.rect)
```

Wir werden print statements an den Anfang jeder Methode stellen und schauen, was in der command line ausgegeben wird.

**Sprite.\_\_init\_\_**  
**Sprite.update**  
**Sprite.draw**

# Observe

```
class Pepper(Sprite):
    def __init__(self, rect, color=RED, speed=10):
        super().__init__(rect, color)
        print("Pepper.__init__")
        self.speed = speed

    def update(self):
        print("Pepper.update")
        self.rect.top = self.rect.top + self.speed

    def touch_ground_update(self, ground_rect):
        print("Pepper.touch_ground_update")
        if self.rect.bottom >= ground_rect.bottom:
            self.rect.top = ground_rect.top
```

Wir werden print statements an den Anfang jeder Methode stellen und schauen, was in der command line ausgegeben wird.

**Pepper.\_\_init\_\_** (after super())

**Pepper.update**

**Pepper.touch\_ground\_update**

# Observe

```
class Player(Sprite):
    def __init__(self, rect, color=BLUE, speed=10):
        super().__init__(rect, color)
        print("Player.__init__")
        self.speed = speed

    def move(self, keystates):
        print("Player.move")
        if keystates[K_LEFT] or keystates[K_a]:
            self.rect.x = self.rect.x - self.speed
        if keystates[K_RIGHT] or keystates[K_d]:
            self.rect.x = self.rect.x + self.speed
```

Wir werden print statements an den Anfang jeder Methode stellen und schauen, was in der command line ausgegeben wird.

**Player.\_\_init\_\_** (after super())

**Player.update**



# Observe

```
pepper = Pepper(pygame.Rect(0, 0, 80, 100))
pepper.rect.midtop = screen_rect.midtop

player = Player(pygame.Rect(0, 0, 80, 80))
player.rect.midbottom = screen_rect.midbottom
quit()
```

Vor der Game Loop erstellen wir ein Pepper-Objekt und ein Player-Objekt und sagen dem Programm dann, dass es stoppen soll.

Dies wird ausgegeben:

Sprite.\_\_init\_\_

Pepper.\_\_init\_\_

Sprite.\_\_init\_\_

Player.\_\_init\_\_

# Observe

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    GAME_SCREEN.fill(WHITE)
    pepper.update()
    keystates = pygame.key.get_pressed()
    player.move(keystates)
    pepper.touch_ground_update(screen_rect)
    pepper.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)
    pygame.display.update()
    quit()
```

Entfernen wir nun `quit()` und führen unsere Game Loop aus. Im Moment verwenden wir nur eine Pepper, keine `pepper_list`. Beendet das Spiel nach einer Iteration.

# Observe

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()

    GAME_SCREEN.fill(WHITE)
    pepper.update()
    keystates = pygame.key.get_pressed()
    player.move(keystates)
    pepper.touch_ground_update(screen_rect)
    pepper.draw(GAME_SCREEN)
    player.draw(GAME_SCREEN)
    pygame.display.update()
    quit()
```

Dies wird ausgegeben:

Sprite.\_\_init\_\_

Pepper.\_\_init\_\_

Sprite.\_\_init\_\_

Player.\_\_init\_\_

Pepper.update

Player.move

Pepper.touch\_ground\_update

Sprite.draw

Sprite.draw