

Philipps



Universität  
Marburg

# Advancing in Python and Pygame

Donnerstag

01.09.2022

9:30 – 10:30 Uhr



hessian.AI



# **Skip class inheritance if done on tuesday**



Part 1

# Class inheritance

# 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.

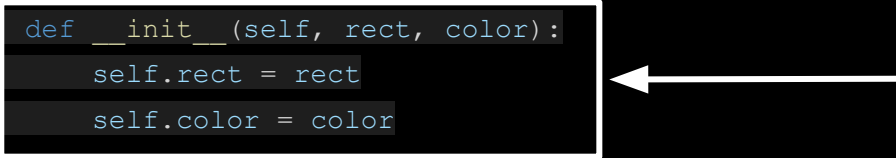


# Step 1: Gemeinsamkeiten in die Sprite-Klasse verschieben

```
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



Part 2

# Pygame classes

# Pygame classes and modules

- Wir müssen nicht immer unsere eigenen Klassen erstellen, da Pygame viele Klassen und Module bereitstellt.
- Hier werden grundlegend sinnvolle Konzepte der Spieleprogrammierung bereitgestellt, die dabei helfen, den Code schneller und sauberer zu machen.
- Da du schon Grundlegendes über Klassen gelernt hast, schauen wir uns einige nützliche tools von Pygame an.
- Du kannst auch immer in die Dokumentation schauen.  
<https://www.pygame.org/docs/>

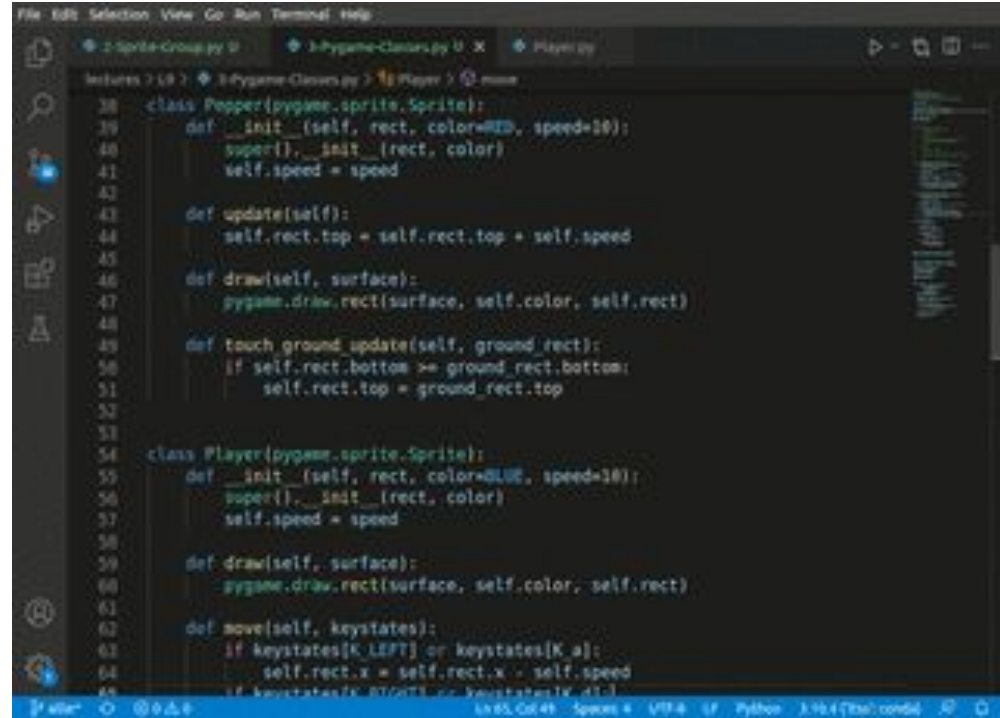
# pygame.sprite.Sprite

- [pygame.sprite.Sprite](#) ist eine einfache Basisklasse für sichtbare Spielobjekte
- Versuchen wir, den Code, an dem wir gerade gearbeitet haben, so zu ändern, dass er diese Klasse anstelle der von uns erstellten Klasse verwendet



# Tipp

In VS Code kann man mit dem Mauszeiger über etwas, das man importiert hat, fahren und eine Vorschau erhalten. Man kann auch darauf klicken, um die Implementierung zu sehen. Anschließend können wir einen Blick darauf werfen, was an unserem Code anders ist.



```
File Edit Selection View Go Run Terminal Help
2-Sprite-Group.py 3-Pygame-Classes.py 4-Player.py
Pepper.py
38 class Pepper(pygame.sprite.Sprite):
39     def __init__(self, rect, color=RED, speed=10):
40         super().__init__(rect, color)
41         self.speed = speed
42
43     def update(self):
44         self.rect.top = self.rect.top + self.speed
45
46     def draw(self, surface):
47         pygame.draw.rect(surface, self.color, self.rect)
48
49     def touch_ground_update(self, ground_rect):
50         if self.rect.bottom >= ground_rect.bottom:
51             self.rect.top = ground_rect.top
52
53
54 class Player(pygame.sprite.Sprite):
55     def __init__(self, rect, color=BLUE, speed=10):
56         super().__init__(rect, color)
57         self.speed = speed
58
59     def draw(self, surface):
60         pygame.draw.rect(surface, self.color, self.rect)
61
62     def move(self, keystates):
63         if keystates[K.LEFT] or keystates[K.a]:
64             self.rect.x = self.rect.x - self.speed
65         if keystates[K.RIGHT] or keystates[K.d]:
```

# Let's get you some starter code...

Code: `lecture_8_codes/2-Pygame-Classes.py`

Starte zunächst das Programm und sieh dir an, was du im Spiel machen kannst.

Dann wollen wir uns ansehen, wie der Code funktioniert.

# Imports, Initialisierung und Konstanten

Am Anfang der Datei gibt es nichts Neues. Diese sind typisch für den Beginn eines Programms

```
""" Imports, initialization, and constants """  
import sys, pygame  
from pygame.locals import *  
  
pygame.init()  
SCREEN_RECT = pygame.Rect(0,0,640, 480)  
WHITE = (255, 255, 255)  
GAME_SCREEN = pygame.display.set_mode((SCREEN_RECT.width, SCREEN_RECT.height))  
pygame.display.set_caption('Pygame Classes and Modules')
```



# Main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Scrolle zum Ende der Datei, um die main-Funktion zu finden.

Hier sollte alles ziemlich vertraut aussehen. Wir haben unser Spielerobjekt, die Uhr und die Game Loop.

Was ist neu?

# Main-Funktion

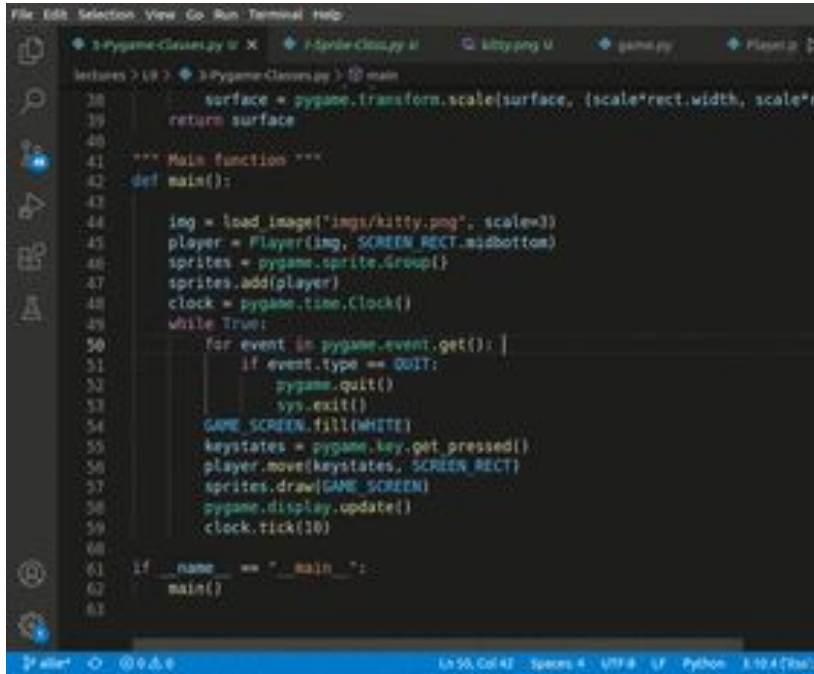
```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Scrolle zum Ende der Datei, um die main-Funktion zu finden.

Hier sollte alles ziemlich vertraut aussehen. Wir haben unser Spielerobjekt, die Uhr und die Game Loop.

Was ist neu? **load\_image()** und **pygame.sprite.Group()**.

# Load image



```
File Edit Selection View Go Run Terminal Help
2-Pygame-Classes.py x 1-Sprite-Class.py 1-kitty.png 1-game.py 1-Player.py
lectures > 19 > 2-Pygame-Classes.py > 10 main
38 surface = pygame.transform.scale(surface, (scale*rect.width, scale*rect.height))
39 return surface
40
41 """ Main function """
42 def main():
43
44     img = load_image("imgs/kitty.png", scale=3)
45     player = Player(img, SCREEN_RECT.midbottom)
46     sprites = pygame.sprite.Group()
47     sprites.add(player)
48     clock = pygame.time.Clock()
49     while True:
50         for event in pygame.event.get():
51             if event.type == QUIT:
52                 pygame.quit()
53                 sys.exit()
54             GAME_SCREEN.fill(WHITE)
55             keystates = pygame.key.get_pressed()
56             player.move(keystates, SCREEN_RECT)
57             sprites.draw(GAME_SCREEN)
58             pygame.display.update()
59             clock.tick(10)
60
61 if __name__ == "__main__":
62     main()
63
```

Werfen wir einen Blick auf die Funktion `load_image`. In VS Code können wir mit **Strg+Klick** auf den Namen der Funktion klicken und gelangen so zu ihrer Definition.

Sie befindet sich direkt über der Hauptfunktion :)

# Load image

```
""" Load image function """
def load_image(file_path, size=None, scale=None):
    """
    size: tuple (width, height)
    scale: integer value to scale the image
    """
    surface = pygame.image.load(file_path)
    if size:
        surface = pygame.transform.scale(surface, size)
    if scale:
        rect = surface.get_rect()
        surface = pygame.transform.scale(surface, (scale*rect.width, scale*rect.height))
    return surface
```

Wenn du ein Bild für deine Sprites verwenden willst, kannst du einfach `pygame.image.load()` verwenden und den Pfad zu deiner Bilddatei übergeben.

# Load image

```
""" Load image function """
def load_image(file_path, size=None, scale=None):
    """
    size: tuple (width, height)
    scale: integer value to scale the image
    """
    surface = pygame.image.load(file_path)
    if size:
        surface = pygame.transform.scale(surface, size)
    if scale:
        rect = surface.get_rect()
        surface = pygame.transform.scale(surface, (scale*rect.width, scale*rect.height))
    return surface
```

Wir wollten die Möglichkeit haben, die Größe oder Skalierung des Bildes zu ändern, also haben wir unsere eigene Funktion mit einigen zusätzlichen Schritten entwickelt, um das zu tun. Wie funktioniert dies?

# Main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Kehren wir nun zur main-Funktion zurück und sehen uns an, wie wir die Funktion `load_image()` verwenden.

# Main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Wir geben den Pfad zu unserem Bild an, und zwar entsprechend der Position, an der wir den Code ausführen.

Außerdem geben wir einen Wert für scale an, um der Funktion mitzuteilen, dass das Bild dreimal so groß wie seine Originalgröße sein soll.

# Main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Dann nehmen wir die Ausgabe von `load_image()` und übergeben sie an unsere Initialisierung des Objekts der Klasse `Player`.

Wechseln wir nun zum Code der `Player`-Klasse, um zu sehen, was die Klasse mit dem Bild macht.



# Player class

```
class Player(pygame.sprite.Sprite):  
    def __init__(self, image, midbottom, speed=10):  
        super().__init__()  
        self.image = image  
        self.rect = self.image.get_rect(midbottom=midbottom)  
        self.speed = speed  
  
    def move(self, keystates, boundaries):  
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
        y_dir = keystates[K_DOWN] - keystates[K_UP]  
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
        self.rect.clamp_ip(boundaries)
```

Dies ist der  
gesamte Code  
unserer  
Player-Klasse.

# pygame.sprite.Sprite

```
class Player(pygame.sprite.Sprite):  
    def __init__(self, image, midbottom, speed=10):  
        super().__init__()   
        self.image = image  
        self.rect = self.image.get_rect(midbottom=midbottom)  
        self.speed = speed  
  
    def move(self, keystates, boundaries):  
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
        y_dir = keystates[K_DOWN] - keystates[K_UP]  
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
        self.rect.clamp_ip(boundaries)
```

Beachte, dass  
Player eine  
pygame-Elternklas  
se hat.

# Player class

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Die Klasse hat ein Attribut "image", dem das Bild zugewiesen wird, das wir ihr in der main-Funktion übergeben haben.

# Player class

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Wie üblich brauchen wir unser imaginäres Rechteck, um zu wissen, wo sich der Spieler auf dem Bildschirm befindet.

# Player class

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Aber dieser Teil  
sieht ein wenig  
anders aus.

# pygame.Surface.get\_rect()

Das Bild ist ein **pygame.Surface** Objekt.

```
self.image.get_rect(midbottom=midbottom
```

pygame.Surface hat eine Methode **get\_rect()**, die die rechteckige Fläche des Oberflächenobjekts zurückgibt.



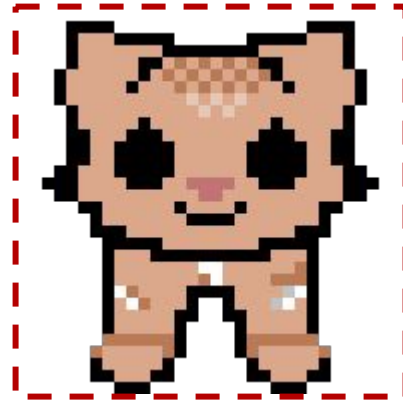
# pygame.Surface.get\_rect()

Das Bild ist ein **pygame.Surface** Objekt.

```
self.image.get_rect(midbottom=midbottom
```

pygame.Surface hat eine Methode **get\_rect()**, die die rechteckige Fläche des Oberflächenobjekts zurückgibt.

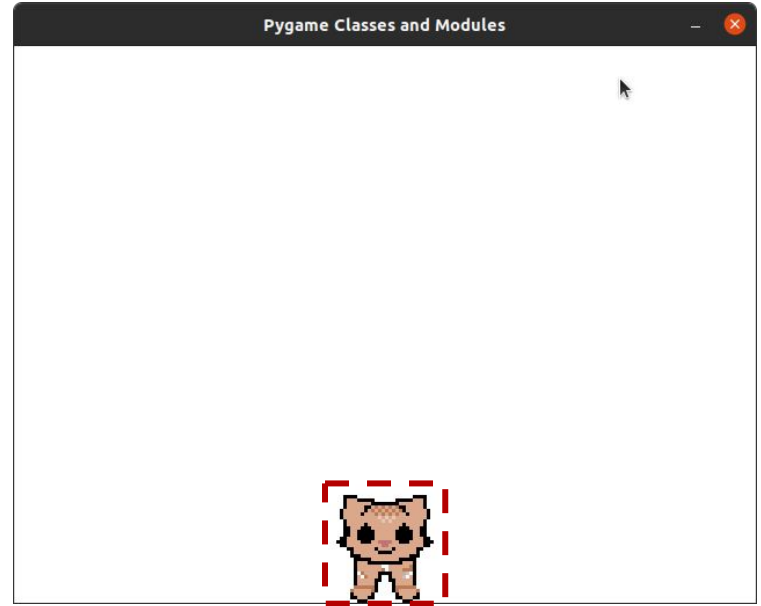
Damit wollen wir also unser imaginäres Rechteck um unser Sprite-Bild erhalten.



# pygame.Surface.get\_rect()

Dank der Implementierung von `pygame.Surface.get_rect()` können wir den Startpunkt unseres Sprites festlegen, indem wir die gewünschte (x,y) Koordinate mit einem Schlüsselwort, einem der `pygame.Rect` Attribute, übergeben.

```
self.image.get_rect(midbottom=midbottom
```

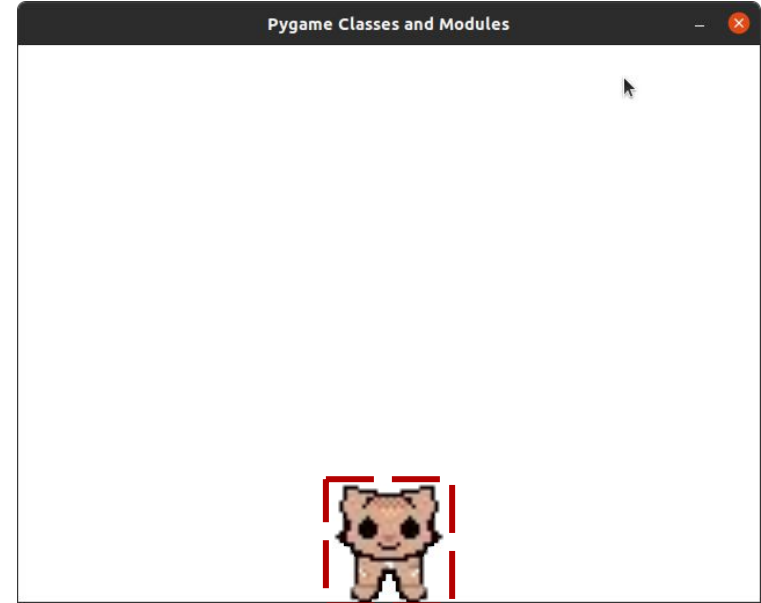




# pygame.Surface

Vergiss nicht, dass du in der Dokumentation immer mehr darüber erfahren kannst, was du mit pygame.Surface machen kannst!

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



# Player class

```
class Player(pygame.sprite.Sprite):  
    def __init__(self, image, midbottom, speed=10):  
        super().__init__()   
        self.image = image  
        self.rect = self.image.get_rect(midbottom=midbottom)  
        self.speed = speed  
  
    def move(self, keystates, boundaries):  
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
        y_dir = keystates[K_DOWN] - keystates[K_UP]  
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
        self.rect.clamp_ip(boundaries)
```

Damit haben wir eine gute Möglichkeit, unser imaginäres Rechteck und seine Position festzulegen.

# Main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Zurück zur Main-Funktion.

Was ist dieses "sprites" Objekt?

Es ist ein `pygame.sprite.Group` Objekt.

Documentation ;)

<https://www.pygame.org/docs/ref/sprite.html#pygame.sprite.Group>

Schauen wir uns zunächst an, was wir mit diesem Objekt in der main-Funktion machen.

# main-Funktion

## 1. Wir fügen den Player hinzu

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

1. Wir fügen den Player hinzu. Das sagt uns, dass es sich um eine Art Container für Sprite-Objekte handeln könnte.

# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

1. Wir fügen den Player hinzu. Das sagt uns, dass es sich um eine Art Container für Sprite-Objekte handeln könnte.
2. Dann rufen wir `sprites.draw()` auf.

# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

1. Wir fügen den Player hinzu. Das sagt uns, dass es sich um eine Art Container für Sprite-Objekte handeln könnte.
2. Dann rufen wir `sprites.draw()` auf. **Erinnert euch das an etwas?**



# pygame.sprite.Group

```
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)
```

Pygame bietet ähnliche Klassen wie unsere PepperGroup, die wir in einem früheren Abschnitt erstellt haben. Wir müssen also nicht unsere eigene Klasse erstellen!

Documentation:

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



# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

- Zurzeit hat die Sprite Group nur ein Sprite: den Spieler.
- Wenn wir dem Spiel weitere Objekte hinzufügen wollen, können wir ein weiteres `pygame.sprite.Sprite`-Objekt erstellen und es der Sprite-Group hinzufügen.
- Dann wird das Objekt gezeichnet.
- Wenn wir ein Verhalten hinzufügen wollen, wie z.B. das Fallen, wenn die Zeit vergeht, was müssen wir dann zur main-Funktion hinzufügen?

# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
        GAME_SCREEN.fill(WHITE)  
        keystates = pygame.key.get_pressed()  
        player.move(keystates, SCREEN_RECT)  
        sprites.draw(GAME_SCREEN)  
        pygame.display.update()  
        clock.tick(10)
```

Wenn wir ein Verhalten hinzufügen wollen, wie z.B. das Fallen, wenn die Zeit vergeht, was müssen wir dann zur main-Funktion hinzufügen?

Wir müssen `sprites.update()` aufrufen. Außerdem müssen wir die Update-Methode des neuen Objekts implementieren.

# main-Funktion

```
def main():  
    img = load_image("imgs/kitty.png", scale=3)  
    player = Player(img, SCREEN_RECT.midbottom)  
    sprites = pygame.sprite.Group()  
    sprites.add(player)  
    clock = pygame.time.Clock()  
    while True:  
        for event in pygame.event.get():  
            if event.type == QUIT:  
                pygame.quit()  
                sys.exit()  
            GAME_SCREEN.fill(WHITE)  
            keystates = pygame.key.get_pressed()  
            player.move(keystates, SCREEN_RECT)  
            sprites.draw(GAME_SCREEN)  
            pygame.display.update()  
            clock.tick(10)
```

Nun zum Player.move.

In der main-Funktion holen wir uns die Tastenzustände und übergeben sie an Player.move, wie wir es in unseren vorherigen Tutorials getan haben.

Wir übergeben SCREEN\_RECT. Was machen wir innerhalb dieser Methode?

# Player.move

```
class Player(pygame.sprite.Sprite):  
    def __init__(self, image, midbottom, speed=10):  
        super().__init__()  
        self.image = image  
        self.rect = self.image.get_rect(midbottom=midbottom)  
        self.speed = speed  
  
    def move(self, keystates, boundaries):  
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
        y_dir = keystates[K_DOWN] - keystates[K_UP]  
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
        self.rect.clamp_ip(boundaries)
```

Player.move hat zwei  
Parameter: Keystates und  
boundaries.

# Player.move

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

In diesem Spiel können wir den Spieler rund um den Bildschirm bewegen, nicht nur von links nach rechts.

# Player.move

```
class Player(pygame.sprite.Sprite):  
    def __init__(self, image, midbottom, speed=10):  
        super().__init__()   
        self.image = image  
        self.rect = self.image.get_rect(midbottom=midbottom)  
        self.speed = speed  
  
    def move(self, keystates, boundaries):  
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
        y_dir = keystates[K_DOWN] - keystates[K_UP]  
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
        self.rect.clamp_ip(boundaries)
```

Obwohl die Implementierung etwas anders aussieht, ist die Funktionalität dieselbe, die ihr bereits kennt.

# Player.move

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

pygame.Rect hat tatsächlich diese move-Methode. Du kannst die Änderung in x und die Änderung in y übergeben. Es wird ein Rechteck mit der geänderten Position zurückgegeben.

# Player.move: Exercise.

```
def move(self, keystates, boundaries):  
    x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
    y_dir = keystates[K_DOWN] - keystates[K_UP]  
    self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
    self.rect.clamp_ip(boundaries)
```

1. Welchen Wert hat **x\_dir**, wenn nur die **rechte Pfeiltaste** gedrückt wird?
2. Welchen Wert hat **x\_dir**, wenn nur die **linke Pfeiltaste** gedrückt wird?
3. Welchen Wert hat **x\_dir**, wenn sowohl die **linke als auch die rechte Pfeiltaste** gedrückt werden?
4. Wie hoch muss der Wert von **y\_dir** sein, damit der Spieler auf dem Bildschirm nach oben gehen kann?



# Player.move: Solution.

```
def move(self, keystates, boundaries):  
    x_dir = keystates[K_RIGHT] - keystates[K_LEFT]  
    y_dir = keystates[K_DOWN] - keystates[K_UP]  
    self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)  
    self.rect.clamp_ip(boundaries)
```

1. Welchen Wert hat **x\_dir**, wenn nur die **rechte Pfeiltaste** gedrückt wird? **1**
2. Welchen Wert hat **x\_dir**, wenn nur die **linke Pfeiltaste** gedrückt wird? **-1**
3. Welchen Wert hat **x\_dir**, wenn sowohl die **linke als auch die rechte Pfeiltaste** gedrückt werden? **0**
4. Wie hoch muss der Wert von **y\_dir** sein, damit der Spieler auf dem Bildschirm nach oben gehen kann? **-1**

# Player.move

```
class Player(pygame.sprite.Sprite):
    def __init__(self, image, midbottom, speed=10):
        super().__init__()
        self.image = image
        self.rect = self.image.get_rect(midbottom=midbottom)
        self.speed = speed

    def move(self, keystates, boundaries):
        x_dir = keystates[K_RIGHT] - keystates[K_LEFT]
        y_dir = keystates[K_DOWN] - keystates[K_UP]
        self.rect = self.rect.move(x_dir * self.speed, y_dir * self.speed)
        self.rect.clamp_ip(boundaries)
```

Zum Schluss rufen wir `self.rect.clamp_ip()` auf und geben die boundaries ein. Überlege dir, wie sich der Spieler verhält, wenn du dich auf dem Bildschirm bewegst. Was denkst du, was dies bewirkt?