Inhaltsverzeichnis

Aktualisierung

Die Pygame Tools sind einer stetigen Weiterentwicklung unterworfen. Es ist wichtig, dass immer die aktuellste Version benutzt wird.

Die Pygame Tools werden in einer Videoserie auf Youtube verwendet. Ich gebe mir Mühe, dass auch die alten Beispiele mit der neusten Version der Tools ausführbar bleiben. In jedem Projekt befindet sich eine Version von Lib.zip, die der im Projekt verwendeten Version entspricht. Es sollte aber nicht notwendig sein, auf diese zurückzugreifen.

https://www.youtube.com/playlist?list=PL4dxj1rGc3b0UH2asMyiN7HAd_l-s7io_

Die aktuellste Version ist auf Github zu finden.

https://github.com/hobbyelektroniker/SpassMitPygame

Dabei sollten die Dateien ***Lib.zip***, ***Vorlage.zip*** und dieses Dokument (***PygameToolsZusammenfassung.pdf***) gleichzeitig aktualisiert werden.

Diese Zusammenfassung ist kein richtiges Handbuch. Es ist lediglich eine Zusammenfassung der wichtigsten Funktionen. Wie die Tools verwendet werden, wird in den Videos erklärt.

Hilfsfunktionen, unabhängig von Pygame

register_object()

```
register_object(name, obj, overwrite=False, no_error=False)
```

Ein beliebiges Objekt (***obj***) wird mit dem Namen ***name*** global registriert.
Ein Name kann nur einmal registriert werden. Ein Überschreiben ist nur möglich, wenn ***overwrite*** angegeben wird. Falls ***no_error*** angegeben ist erzeugt der Versuch eines unerlaubten Überschreibens keine Fehlermeldung.

get_object()

```
obj = get_object(name, default=None)
```

Das Objekt mit dem Namen ***name*** wird zurückgegeben. Wenn der Name nicht registriert ist, wird der Wert von ***default*** zurückgegeben.

unregister_object()

```
unregister_object(name)
```

Das Objekt mit dem Namen ***name*** wird aus der Registrierung entfernt. Wenn der Name nicht existiert, erfolgt keine Fehlermeldung.

Hilfsklassen, unabhängig von Pygame

Record

Diese Klasse implementiert einen einfachen Recordtyp mit vordefinierten Feldern.
Jedem Feld kann ein Standardwert zugeordnet werden.
Die Struktur kann mit print() und repr() sinnvoll ausgelesen werden.
repr() kann zusammen mit ***eval()*** zur Erzeugung einer Kopie verwendet werden.

Erzeugen und erweitern

```
r = Record(*args, **kwargs)
```

Die Felder können mit oder ohne Vorgabewerte definiert werden.
Feldnamen ohne Vorgabewerte können mit dem Wert None oder als String angegeben werden.

```
r = Record('a', b=None, c='C', d=5)
```

Zusätzliche Felder können auch nachträglich hinzugefügt werden.

```
r.neues_feld = 25
```

Alternativ kann ein Record auch aus einem Dictionary erstellt werden.

```
dict = {
    'a': None,
    'b': None,
    'c': 'C',
    'd': 5
}
r = Record.from_dict(dict)
```

Arbeiten mit dem Record

Die Felder können einfach mit der Punktschreibweise angesprochen werden.

```
r.b = 'B'
print(r.c)
```

Der Record kann aber auch als Dictionary behandelt werden.

```
dict = r.as_dict()
dict['c'] = 'Ein neuer Wert'
```

Es handelt sich bei dict um dieselbe Instanz wie r.
Um eine Kopie zu erhalten, muss `dict = r.as_dict().copy()` verwendet werden.

Ausgabe mit print() und print(repr())

```
r = Record('a', b=None, c='C', d=5)
print(r)
```
a: None
b: None
c: "C"
d: 5

```
print(repr(r))
```
Record('a', 'b', c="C", d=5)

repr(r) kann zum Erzeugen einer Kopie des Records verwendet werden.
```
r1 = eval(repr(r))
```

TraceVar

TraceVar implementiert eine überwachte Variable.
Sie kann bei Schreib-, Lese- oder Änderungsoperationen Funktionen aufrufen.

Erzeugen

```
v = TraceVar(25)
```

erzeugt eine Variable mit dem Initialwert 25. Die Angabe des Startwertes ist optional, wird aber dringend empfohlen.

Wert ändern oder abfragen

```
v.set(val)
v.value = val
val = v.get()
val = v.value
```

Funktionen verwalten

Wenn ihr Wert geschrieben, gelesen oder verändert wird, kann die Tracevariable eine oder mehrere Funktionen aufrufen. Eine aufrufbare Funktion muss immer das Argument ***tag*** entgegennehmen können.

```
def func(tag=None):
    ...
```

Bei den folgenden Aufrufen wird ***tag*** None gesetzt.

```
v.trace_add(fn)  # Wird aufgerufen, wenn die Variable geschrieben wird.
v.trace_add(fn, 'w')  # entspricht v.trace_add(fn).
v.trace_add(fn, 'r')  # Wird aufgerufen, wenn die Variable gelesen wird.
v.trace_add(fn, 'c')  # Wird aufgerufen, wenn die Variable geändert wird.
```

tag kann definiert werden.

```
v.trace_add(fn, 'r', tag='Zusatzinfo')  # Es wird fn('Zusatzinfo') aufgerufen.
```

Funktionen können auch wieder entfernt werden.

```
v.trace_clear()      # Löscht alle Funktionen.
v.trace('w')         # Löscht alle 'w' - Funktionen.
v.trace_clear('rwc') # Löscht alle 'r', 'w' und 'c' Funktionen.
# Es sind alle Kombinationen in beliebiger Reihenfolge erlaubt.

v.trace_remove(fn)      # Löscht die 'w' - Funktion.
v.trace_remove(fn, 'w') # Entspricht v.trace_remove(fn).
v.trace_remove(fn, 'c') # Löscht die 'c' - Funktion.
v.trace_remove(fn, 'r') # Löscht dir 'r' - Funktion.
```

Wenn ***tag*** verwendet wurde, muss dieser korrekt angegeben werden.

```
v.trace_remove(fn, 'r', tag='Zusatzinfo')
```

Aufrufen der Funktionen

Normalerweise werden die Funktionen automatisch aufgerufen. In Spezialfällen kann es aber notwendig sein, eine Funktion direkt aufzurufen.

```
v.trace()       # Alle Funktionen aufrufen
v.trace('w')    # Alle 'w' - Funktionen aufrufen.
v.trace('rwc')  # Alle 'r', 'w' und 'c' Funktionen aufrufen.
# Es sind alle Kombinationen in beliebiger Reihenfolge erlaubt.
```

Hilfsklassen, abhängig von Pygame

Size

Diese Klasse ist von der Pygame - Klasse Vector2 abgeleitet und verwaltet eine zweidimensionale Grösse mit Breite und Höhe.
Objekte dieser Klasse können an Pygame - Funktionen übergeben werden, die ein Tupel mit Breite und Höhe erwarten.

Erzeugen

```
sz = Size(w, h=None)
```

Wenn h None ist, wird dafür der Wert von w eingesetzt.

w: Breite, auch als .x verfügbar
h: Höhe, auch als .y verfügbar

Ändern und abfragen

```
sz.w = 25
hoehe = sz.h
```

Position

Diese Klasse ist von der Pygame - Klasse Vector2 abgeleitet und verwaltet eine Position.
Objekte dieser Klasse können an Pygame - Funktionen übergeben werden, die ein Tupel mit x- und y-Koordinaten erwarten.

Erzeugen

```
pos = Position(x, y)
```

Direkte Angabe der beiden Koordinatenpunkte.

```
pos = Position.from_deg(deg, length=1, center=(0, 0))
```

Die Position wird definiert durch einen Vektor mit Zentrum, Winkel und einer Länge.
0 Grad entspricht einem Vektor nach rechts.

```
pos = Position.from_rad(rad, length=1, center=(0, 0))
```

Die Position wird definiert durch einen Vektor mit Zentrum, Winkel im Bogenmass und einer Länge.
0 entspricht einem Vektor nach rechts.

Ändern und abfragen

Alle Properties können lesend und schreibend verwendet werden.
.x: x - Koordinate
.y: y - Koordinate
.length: Länge des Vektors
.deg: Winkel des Vektors
.rad: Winkel im Bogenmass

Direction
Diese Klasse ist von der Pygame - Klasse Vector2 abgeleitet und gibt eine Richtung an. Es handelt sich dabei um einen Vektor mit der Länge 1.

Erzeugen
```
dir = Direction(x=0, y=0, deg=None, rad=None)

dir = Direction()
```
Dies ist die Ausnahme, da der Vektor die Länge 0 hat. Er wird als 'keine Richtung' interpretiert.

```
dir = Direction(x=1)
```
Zeigt nach rechts

```
dir = Direction(1, 1)
```
Zeigt nach unten rechts.

```
dir = Direction(deg=270)
```
Zeigt nach oben

```
dir = Direction(rad=π)
```
Zeigt nach links

Ändern und abfragen
```
dir = Direction() # Variable erzeugen (keine Richtung)
dir.left()        # Nach links
dir.right()       # Nach rechts
dir.up()          # Nach oben
dir.down()        # Nach unten
dir.clear()       # Keine Richtung
```

Alle Properties können gelesen und geschrieben werden.
`.rad`: Winkel im Bogenmass
`.deg`: Winkel in Grad
`.x`: x - Richtung
`.y`: y - Richtung

`x` und `y` sollten nicht direkt geschrieben werden, da dabei keine Normalisierung auf Länge 1 erfolgt.

Function
Funktionsobjekt für Funktionen, die im Hintergrund ablaufen.

Erzeugen
Ausschliesslich durch ***game.function()*** oder ***game_object.function()***.

Eigenschaften und Funktionen

seconds	Intervall in Sekunden
loops	Anzahl Aufrufe. None = nicht begrenzt.
kwargs	Zusätzliche benannte Argumente
stop()	Beendet die Aufrufe

Media
Diese Klasse erleichtert die Arbeit mit Bild- und Audiodateien.

Erzeugen
Von dieser Klasse wird keine Instanz erstellt. Der Zugriff erfolgt ausschliesslich über Klassenmethoden.

Bilddateien
Mit ***load_image*** kann eine Bilddatei geladen werden.

```
img = Media.load_image(filename, size=None, width=None, height=None,
                       colorkey=None, createfullname=True)
```

`load_image('bild.png')` lädt eine Datei aus dem Media - Verzeichnis des Projekts. Falls eine Datei aus einem anderen Verzeichnis geladen werden muss, kann als Filename der vollständige Pfad angegeben werden. Dazu muss ***createfullname*** auf False gesetzt werden.

Das Bild kann beim Laden in seiner Grösse angepasst werden. ***size*** legt Höhe und Breite fest. Da dabei Verzerrungen auftreten können, ist es ratsam stattdessen mit ***width*** oder ***height*** zu arbeiten. Dabei sollte nur einer der Werte gesetzt werden, der andere passt sich dann automatisch so an, dass das Seitenverhältnis des Bildes gewahrt bleibt.

Einige Bilddateien bieten direkt einen transparenten Hintergrund. Bei anderen wird eine bestimmte Farbe als Hintergrund verwendet. Diese kann als RGB - Tupel über ***colorkey*** übergeben werden. Oft ist dieser Wert aber nicht bekannt. So kann auch -1 an ***colorkey*** übergeben werden. Dann wird die Farbe des ersten Punktes links oben als Tranparenzfarbe verwendet.

Mit ***load_images*** können mehrere Bilddateien geladen werden. Es wird dann eine Liste von Images zurückgegeben. Das wird oft gebraucht, wenn die Bilddateien eine Animation bilden sollen. Das Laden der Animation erfolgt normalerweise über über ***load_animation*** von ***AnimatedGameObject***.

```
imgs = Media.load_images(filename, size=None, width=None, height=None,
                         colorkey=None, createfullname=True)
```

filename muss dabei mit * angegeben werden.
`load_images('bild*.png')`

Musik
Es kann immer nur ein Musikstück geladen und abgespielt werden.

```
Media.play_music(filename)
Media.set_volume(volume)
Media.pause_music()
Media.unpause_music()
Media.stop_music()
```

Sounds
Sounds werden geladen und mit einem Namen versehen. Sie können danach beliebig abgespielt werden.

```
Media.load_sound(name, filename)
Media.set_soundvolume(volume)       # generelle Lautstärke (0 bis 1.0)
Media.play_sound(name, volume=None) # optional individuelle Lautstärke
```

Hilfsfunktionen, abhängig von Pygame

hitbox_collide()
```
hitbox_collide(sprite, other) -> bool
```

Testet, ob ein Sprite mit einem anderen kollidiert. Dabei werden Korrekturen an den Hitboxen berücksichtigt.

set_rect_pos()
```
set_rect_pos(rect: Rect, pos, anchor='tl')
```

Setzt in ***rect*** die Position entsprechend ***pos*** und ***anchor***.

get_rect_pos()
```
get_rect_pos(rect: Rect, anchor='tl') -> Position
```

Ermittelt ***pos*** aus ***anchor*** und ***rect***.

get_size()
```
get_size(surface=None) -> Size
```

get_width()
```
get_width(surface=None) -> float
```

get_height()
```
get_height(surface=None) -> float
```

get_center()
```
get_center(surface=None) -> Position
```

Wenn ***surface*** nicht angegeben wird, bezieht sich das Resultat auf den gesamten ***screen*** des Games.

get_mouse_pos()
```
get_mouse_pos() -> Position
```

create_text()
```
create_text(text, font, antialias=True, color='white', background=None)
```
gibt ein surface mit dem gerenderten Text zurück.

draw_text()
```
draw_text(surface, text, pos, font, antialias=True, color='white',
          background=None, anchor='tl')
```
zeichnet einen Text direkt in das ***surface***.

Hauptklassen

Game

Das ist die Basisklasse jedes Games. Das eigentliche Game wird immer mit einer abgeleiteten Klasse erzeugt. Diese Basisklasse stellt aber bereits die wichtigsten Funktionen zur Verfügung.

Erzeugen

```
game = Game(title=" ", width=800, height=600, fps=60, fontsize=30,
            bgcolor='black')
```

title	Titel, der in der Kopfzeile des Fensters angezeigt wird.
width	Breite in Pixel
height	Höhe in Pixel
fps	Anzahl Frames pro Sekunde
fontsize	Grösse des Standardfonts
bgcolor	Farbe des Hintergrundes

Automatisch bereitgestellte Eigenschaften

screen	Das ganze Spielfeld
background	Der Hintergrund in Grösse des ganzen Spielfelds. Er ist mit der Hintergrundfarbe ***bgcolor*** gefüllt. background wird automatisch als unterste Ebene des Bildschirms gezeichnet.
font	Ein Standardfont in der Grösse ***fontsize***.
visible_sprites	Gruppe der sichtbaren Sprites. Diese werden automatisch gezeichnet.
update_sprites	Gruppe der Sprites, bei denen ***update()*** aufgerufen wird
player_sprites	Gruppe der Spielersprites
other_sprites	Gruppe der anderen Spielobjekte

Fixe Methoden
Diese Methoden sollten nicht überschrieben werden.

show_boxes(*groups)	Zeige die Hitboxen der angegebenen Gruppen.
create_group() -> Group	Erzeuge eine neue Spritegruppe.
create_text(text, font=None, antialias=False, color='white', background=None) -> Surface	Rendert einen Text. Wenn kein Font angegeben ist, wird der Standardfont verwendet. Wenn kein Hintergrund angegeben wird, ist dieser transparent.
function(seconds, func, loops=None, **kwargs) -> Function	Ruft die Funktion ***func*** alle ***seconds*** Sekunden auf. Wenn ***loops*** angegeben ist, stoppen die Anrufe nach den angegebenen Anzahl Aufrufen. Es können beliebig viele benannte Argumente mitgegeben werden. Das Funktionsobjekt wird zurückgegeben. Damit kann man das Verhalten nachträglich beeinflussen. Die Aufrufe laufen im Hintergrund und stören den restlichen Programmablauf nicht.
after(seconds, func, **kwargs)	Nach ***seconds*** Sekunden wird die Funktion ***func*** einmal aufgerufen. Es können beliebig viele benannte Argumente mitgegeben werden. Der Aufrufe wird im Hintergrund ausgelöst und stört den restlichen Programmablauf nicht.
stop()	Beendet das Programm.
run()	Startet das Programm

Methoden zum Erweitern im eigenen Game
Diese Methoden können in der abgeleiteten Gameklasse erweitert werden. Es muss aber immer mit super() die Methode der Basisklasse aufgerufen werden.

__init__(...)	Neue Elemente sollten im Constructor angelegt werden.
activate()	Weitere Initialisierungen, die erst nach der vollständigen Abarbeitung des Constructors durchgeführt werden dürfen.
handle_event(event)	Behandlung eines Pygame - Events. Dies betrifft hauptsächlich Maus und Tastatur.
update(dt)	Berechnung des nächsten Schritts. ***dt*** ist die Zeit in Millisekunden, die seit dem letzten Aufruf von ***update()*** vergangen ist.
draw()	Zeichnen aller Elemente in ***self.screen***. Im Anschluss wird der aktualisierte ***screen*** angezeigt.

SpriteObject

Das ist die Basisklasse für aktiven Elemente eines Spiels. Es handelt sich dabei um erweiterte Pygame - Sprites.

Erzeugen

Üblicherweise wird von dieser Klasse keine Instanz erstellt. Die eigentlichen Spielelemente sind direkt oder indirekt von dieser Basisklasse abgeleitet und erben daher dessen Eigenschaften.

```
sprite = SpriteObject(game, pos=(0, 0), size=(50, 50), anchor='tl',
                      active=True, visible=True,
                      tcolor=(0, 0, 0), group=None, update_sprites=None,
                      visible_sprites=None)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ('tl').
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***visible_sprites***.
tcolor	Transparenzfarbe
group	Zusätzliche optionale Gruppe
update_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.update_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.
visible_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.visible_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.

Automatisch bereitgestellte Eigenschaften

game	Das Game
tcolor	Transparenzfarbe
size	Grösse (Size)
image	Das angezeigte Bild.
rect	Das umhüllende Rechteck von image (Rect aus Pygame)
active	Update wird ausgeführt (in Gruppe update_sprites)
visible	Ist sichtbar (in Gruppe visible_sprites)
anchor	Die Ankerposition
pos	Die aktuelle Position unter Berücksichtigung des Ankers.

Fixe Methoden
Diese Methoden sollten nicht überschrieben werden.

shrink_box(left=None, right=None, top=None, bottom=None)	Verkleinern der Hitbox. Die Box wird an der angegebenen Seite jeweils um die Anzahl Pixel verkleinert.
activate_events(value=True)	Alle Events vom Game werden an handle_event weitergeleitet.
add_var(var)	Hinzufügen einer Tracevariablen.
remove_var(var)	Entfernen einer Tracevariablen.
hitbox() -> Rect	Positonsrechteck der Hitbox.
collide_with_group(group, dokill=False)	Testet auf Kollision mit Elementen einer Gruppe. Gibt eine Liste aller berührten Objekte zurück. Korrigierte Hitboxen werden berücksichtigt. Mit ***dokill*** kann die unmittelbare Zerstörung der berührten Objekte ausgelöst werden.
function(seconds, func, loops=None, **kwargs) -> Function	Ruft die Funktion ***func*** alle ***seconds*** Sekunden auf. Wenn ***loops*** angegeben ist, stoppen die Anrufe nach den angegebenen Anzahl Aufrufen. Es können beliebig viele benannte Argumente mitgegeben werden. Das Funktionsobjekt wird zurückgegeben und kann das Verhalten nachträglich beeinflussen. Die Aufrufe laufen im Hintergrund und stören den restlichen Programmablauf nicht.
after(seconds, func, **kwargs)	Nach seconds Sekunden wird die Funktion ***func*** einmal aufgerufen. Es können beliebig viele benannte Argumente mitgegeben werden. Der Aufrufe wird im Hintergrund ausgelöst und stört den restlichen Programmablauf nicht.
change_anchor(anchor)	Das Element bleibt an seiner Position. ***pos*** wird auf den neuen Anker umgerechnet.
set_anchor(anchor)	***pos*** bleibt erhalten. Das Objekt verschiebt sich so, dass die neue Ankerposition bei ***pos*** liegt.
set_pos(pos)	Die Position wird unter Beibehaltung des Ankers geändert.
set_x(x)	Die x - Position von pos neu festlegen.
set_y(y)	Die y - Position von pos neu festlegen.
move(dx=0, dy=0)	Ändern der Position um die angegebenen Werte. ***pos*** und ***rect*** werden dabei angepasst.
add(*groups)	Hinzufügen zu Gruppen.
remove(*groups)	Entfernen aus Gruppen.
kill()	Entfernen aus allen Gruppen.
alive() -> bool	Gehört das Element noch zu einer Gruppe?

Methoden zum Erweitern in der abgeleiteten Klasse
Diese Methoden können in der abgeleiteten Klasse erweitert werden. Es muss aber immer mit super() die Methode der Basisklasse aufgerufen werden.

__init__(...)	Neue Elemente sollten im Constructor angelegt werden.
handle_event(event)	Behandlung eines Pygame - Events. Dies betrifft hauptsächlich Maus und Tastatur. Die Methode muss vorher mit ***activate_events()*** freigeschaltet werden.
update(dt)	Berechnung des nächsten Schritts. ***dt*** ist die Zeit in Millisekunden, die seit dem letzten Aufruf von ***update()*** vergangen ist.

GameObject

Das ist die Basisklasse für die meisten aktiven Elemente eines Spiels. Sie ist von ***SpriteObject*** abgeleitet. Sie erweitert das ***SpriteObject*** um die Fähigkeit sich zu bewegen.

Erzeugen

Üblicherweise wird von dieser Klasse keine Instanz erstellt. Die eigentlichen Spielelemente werden von dieser Klasse abgeleitet und erben daher die Eigenschaften und die Eigenschaften der Basisklasse ***SpriteObject***.

```
obj = GameObject(game, pos=(0, 0), size=(50, 50), anchor='tl',
                 active=True, visible=True,
                 tcolor=(0, 0, 0), group=None, update_sprites=None,
                 visible_sprites=None)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ('tl').
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***visible_sprites***.
tcolor	Transparenzfarbe
group	Zusätzliche optionale Gruppe
update_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.update_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.
visible_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.visible_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.

Automatisch bereitgestellte Eigenschaften

super().*	Die Eigenschaften von ***SpriteObject***.
speed	Geschwindigkeit in Pixel pro Sekunde (float) in Richtung direction.
direction	Richtung der Bewegung (***Direction***)
gravity	Gravitation, default 200 Sorgt zusammen mit der Masse, dass das Objekt nach unten fällt.
mass	Masse, default 0
impulse	Impuls, der zusammen mit Gravitation und Masse die vertikale Bewegung beeinflusst. Default 0.
gspeed	Durch dir Gravitation ausgelöste Geschwindigkeit nach unten. Berechnet aus mass, impulse und gravity. Eine Bewegung wird nur ausgelöst, wenn ein Boden mit constraints(bottom=yy) definiert ist. Gegen oben kann mit constraints(top=yy) begrenzt werden.

Fixe Methoden
Diese Methoden sollten nicht überschrieben werden.

super().*	Alle entsprechenden Methoden von ***SpriteObject***.
set_image(image)	Ein Bild als Image verwenden. ***pos***, ***rect*** und ***size*** werden automatisch angepasst.
keyboard_move(up=None, down=None, left=None, right=None)	Das Element soll mit der Tastatur gesteuert werden. Alle notwendigen Tasten müssen angegeben werden. Das Element bewegt sich mit der Geschwindigkeit ***speed*** in die entsprechene Richtung bis die Taste losgelassen wird.
constraints(top=None, bottom=None, left=None, right=None)	Begrenzungen, die das Objekt nicht überschreiten kann.
get_constrants()	gibt top, bottom, left und right zurück.

Methoden zum Erweitern in der abgeleiteten Klasse
Diese Methoden können in der abgeleiteten Klasse erweitert werden. Es muss aber immer mit ***super()*** die Methode der Basisklasse aufgerufen werden.

__init__(...)	Neue Elemente sollten im Constructor angelegt werden.
handle_event(event)	Behandlung eines Pygame - Events. Dies betrifft hauptsächlich Maus und Tastatur. Die Methode muss vorher mit ***activate_events()*** freigeschaltet werden.
update(dt)	Berechnung des nächsten Schritts. ***dt*** ist die Zeit in Millisekunden, die seit dem letzten Aufruf von ***update()*** vergangen ist.

AnimatedGameObject

Das ist die Basisklasse für ein animiertes ***GameObject***. Der Unterschied zum normalen GameObject liegt in der animierten Grafik.

Erzeugen

Üblicherweise wird von dieser Klasse keine Instanz erstellt. Die eigentlichen Spielelemente werden von dieser Klasse abgeleitet und erben daher dessen Eigenschaften und die Eigenschaften der Basisklasse GameObject.

```
obj = GameObject(game, pos=(0, 0), size=(50, 50), anchor='tl',
                 active=True, visible=True,
                 tcolor=(0, 0, 0), group=None, update_sprites=None,
                 visible_sprites=None)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ('tl').
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***visible_sprites***.
tcolor	Transparenzfarbe
group	Zusätzliche optionale Gruppe
update_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.update_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.
visible_sprites	Wenn der Wert ***None*** ist, wird das Objekt in die Gruppe ***game.visible_sprites*** eingefügt. Sonst wird die angegebene Gruppe verwendet.

Automatisch bereitgestellte Eigenschaften

super().*	Die Eigenschaften von GameObject.
fps	Geschwindigkeit der Animation.
animating	Property True, False

Fixe Methoden

Diese Methoden sollten nicht überschrieben werden.

super().*	Alle entsprechenden Methoden von ***GameObject***.
load_animation(filename, fps=10, **kwargs)	Laden einer Animation. Als ***kwargs*** können alle Argumente von Media.load_images übergeben werden (size=None, width=None, height=None, colorkey=None, createfullname=True)

start_animating(seconds=None)	Startet die Animation. Falls seconds angegeben ist, wird die Animation nach soviel Sekunden automatisch beendet.
stop_animating()	Stoppt die Animation.

Methoden zum Erweitern in der abgeleiteten Klasse
Diese Methoden können in der abgeleiteten Klasse erweitert werden. Es muss aber immer mit ***super()*** die Methode der Basisklasse aufgerufen werden.

__init__(...)	Neue Elemente sollten im Constructor angelegt werden.
handle_event(event)	Behandlung eines Pygame - Events. Dies betrifft hauptsächlich Maus und Tastatur. Die Methode muss vorher mit ***activate_events()*** freigeschaltet werden.
update(dt)	Berechnung des nächsten Schritts. ***dt*** ist die Zeit in Millisekunden, die seit dem letzten Aufruf von ***update()*** vergangen ist.

Benutzeroberfläche

Label

Ein Label gibt einen Text aus. Der Text kann sehr einfach aktualisiert werden. Die Klasse ist von ***SpriteObject*** abgeleitet.

Erzeugen

```
label = Label(game, pos=(0, 0), value="X", fontsize=50, font=None, anchor='tl',
              color="white", tcolor=(0, 0, 0), var=None, str_format=None,
              visible=True)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ***('tl')***.
var	Hier kann optional eine Tracevariable übergeben werden. Andernfalls wird automatisch eine erstellt und deren Wert auf ***value*** gesetzt.
value	Startwert für eine automatisch erstellte Tracevariable. Wenn ***var*** übergeben wurde, wird ***value*** nicht beachtet.
str_format	Optionale Formatangabe. Beispiel: `var_format="{:3.2f}"`. Ohne diese Angabe wird einfach `str(value)` angezeigt.
font	Wenn ein bereits vorhandener ***font*** angegeben wird, wird dieser verwendet und ***fontsize*** nicht beachtet.
fontsize	Wenn ***font*** nicht angegeben ist, wird ein Standardfont mit der Grösse ***fontsize*** erzeugt.
color	Farbe der Schrift
tcolor	Transparenzfarbe
visible	Das Label ist sichtbar.

Properties

var	Tracevariable (read only)
value	Setzt den Wert der Tracevariablen ***var*** auf ***value***. Dadurch werden alle Aktionen der Tracevariablen ausgelöst.
visible	Das Label ist sichtbar.

Funktionen

render()	Das Label wird neu gezeichnet.

Checkbox

Erzeugen

```
box = Checkbox(game, pos=(0, 0), text="Checkbox", fontsize=50, font=None,
               anchor='tl', color="white", tcolor=(0, 0, 0), func=None,
               tag=None, checked=False, enabled=True, visible=True)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ***('tl')***.
text	Der Text, der hinter der Box angezeigt wird.
font	Wenn ein bereits vorhandener ***font*** angegeben wird, wird dieser verwendet und ***fontsize*** nicht beachtet.
fontsize	Wenn ***font*** nicht angegeben ist, wird ein Standardfont mit der Grösse ***fontsize*** erzeugt.
color	Farbe von Box und Schrift
tcolor	Transparenzfarbe
func	Funktion, die aufgerufen werden soll. def func(box): ...
tag	Optionaler Zusatzinformation. Wenn ***tag*** nicht angegeben ist, wird ***text*** eingesetzt.
checked	Die Box ist angekreuzt.
enabled	Die Box kann angeklickt werden.
visible	Die Checkbox ist sichtbar.

Properties

checked	Die Box ist angekreuzt.
enabled	Die Box kann angeklickt werden.
text	Beschriftung der Checkbox.
visible	Die Checkbox ist sichtbar.

Funktionen

render()	Die Checkbox wird neu gezeichnet.

Slider

Erzeugen

```
slider = Slider(game, pos=(0, 0), size=(100, 10), anchor='tl', color="white",
                bg_color='black', value_range=(0, 100), value=None, var=None,
                tcolor=(0, 0, 0), enabled=True, visible=True, horizontal=True)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ***('tl')***.
size	Grösse des Rechtecks.
color	Farbe des Rechtecks.
bg_color	Hintergrundfarbe
value_range	Minimaler und maximaler Wert.
var	Hier kann optional eine Tracevariable übergeben werden. Andernfalls wird automatisch eine erstellt und deren Wert auf ***value*** gesetzt.
value	Startwert für eine automatisch erstellte Tracevariable. Wenn ***var*** übergeben wurde, wird ***value*** nicht beachtet.
tcolor	Transparenzfarbe
enabled	Der Slider kann betätigt werden.
visible	Der Slider ist sichtbar.
horizontal	Der Slider ist normalerweise auf eine horizontale Betätigung ausgelegt (von links nach rechts). Mit ***horizontal=False*** kann die Betätigung von unten nach oben erfolgen.

Properties

value	Der aktuelle Wert.
enabled	Der Slider kann betätigt werden.
var	Die Tracevariable (readonly).
visible	Die Checkbox ist sichtbar.

Funktionen

render()	Der Slider wird neu gezeichnet.

Button
Schaltfläche mit Text.

Erzeugen
```
button = Button(game, pos=(0, 0), size=(10, 10), text="Button", fontsize=50,
                font=None, anchor='tl', color="white", hot_color="light_blue",
                bg_color='darkblue', tcolor=(0, 0, 0), func=None, tag=None,
                enabled=True, visible=True)
```

game	Das Game wird immer übergeben.
pos	Bestimt in Zusammenarbeit mit ***anchor*** die Position.
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben ***('tl')***.
text	Der Text, der im Button angezeigt wird.
size	Optionale Grösse des Buttons. Wenn ***size*** nicht angegeben wird bestimmt der Text die Grösse.
font	Wenn ein bereits vorhandener ***font*** angegeben wird, wird dieser verwendet und ***fontsize*** nicht beachtet.
fontsize	Wenn ***font*** nicht angegeben ist, wird ein Standardfont mit der Grösse ***fontsize*** erzeugt.
color	Farbe des Textes
bg_color	Farbe des Buttons
hot_color	Farbe des Buttons, wenn die Maus darüber steht.
tcolor	Transparenzfarbe
func	Funktion, die aufgerufen werden soll. def func(button): ...
tag	Optionaler Zusatzinformation. Wenn ***tag*** nicht angegeben ist, wird ***text*** eingesetzt.
enabled	Der Button kann angeklickt werden.
visible	Der Button ist sichtbar.

Properties

text	Text des Buttons. Es erfolgt keine automatische Grössenanpassung des Buttons.
enabled	Der Button kann angeklickt werden.
visible	Der Button ist sichtbar.