Inhaltsverzeichnis

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. Der Versuch eines unerlaubten Überschreibens erzeugt eine Fehlermeldung, falls nicht ***no_error*** angegeben ist.

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 in ***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 ein 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.clear()
Setzt die Richtung auf 'keine Richtung'

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

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

Function
Funktionsobjekt für im Hintergrund ablaufende Funktionen.

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.

pixels_to_move()
```
pixels_to_move(direction: pg.Vector2, speed: float, seconds: float) -> pg.Vector2
```

Berechnet die Verschiebung eines Elementes in Pixeln, abhängig von Geschwindigkeit, Richtung und Zeit.

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

Setzt in ***rect*** die Position entsprechend dem Anker.

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

Ermittelt die Position eines bestimmten Ankers in einem 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) -> float
```

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

Hauptklassen

Game

Das ist die Basisklasse jedes Games. Das eigentliche Game wird immer in 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
update_sprites	Gruppe der Sprites, bei denen update() aufgerufen wird
player_sprites	Gruppe der Spielersprites

Fixe Methoden
Diese Methoden sollten nicht überschrieben werden.

show_boxes(*groups)	Zeige die Hitboxen der angegebenen Gruppen an.
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 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.
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 Sekunden, die seit dem letzten Aufruf von update() vergangen ist.
draw()	Zeichnen aller Elemente in self.screen. Im Anschluss wird das 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 Klasse abgeleitet und erben daher diese Eigenschaften.

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

game	Das Game wird immer übergeben.
pos	Anzeigeposition
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***game.update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***game.visible_sprites***.
tcolor	Transparenzfarbe
group	Optionale Gruppe

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
visible	Ist sichtbar
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

activate_events()	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. 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.
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 Sekunden, die seit dem letzten Aufruf von update() vergangen ist.
draw()	Zeichnen des Elements in self.screen.

GameObject

Das ist die Basisklasse für die meisten aktiven Elemente eines Spiels. Sie ist von ***SpriteObject*** abgeleitet.

Erzeugen

Üblicherweise wird von dieser Klasse keine Instanz erstellt. Die eigentlichen Spielelemente werden von dieser Klasse abgeleitet und erben daher diese 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)
```

game	Das Game wird immer übergeben.
pos	Anzeigeposition
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***game.update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***game.visible_sprites***.
tcolor	Transparenzfarbe
group	Optionale Gruppe

Automatisch bereitgestellte Eigenschaften

super().*	Die Eigenschaften von SpriteObject.
speed	Geschwindigkeit in Pixel pro Sekunde (float).
direction	Richtung der Bewegung (Direction)

Fixe Methoden

Diese Methoden sollten nicht überschrieben werden.

super().*	Alle 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. All notwendigen Tasten müssen angegeben werden. Das Element bewegt sich mit der Geschwindigkeit ***speed*** bis die Taste losgelassen wird.
constraints(top=None, bottom=None, left=None, right=None)	Begrenzungen, die das Objekt nicht überschreiten kann.

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 Sekunden, die seit dem letzten Aufruf von update() vergangen ist.
draw()	Zeichnen des Elements in self.screen.

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 diese Eigenschaften und die Eigenschaften der Basisklasse GameObject.

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

game	Das Game wird immer übergeben.
pos	Anzeigeposition
size	Grösse
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
active	Bei aktiven Objekten wird automatisch ***update()*** aufgerufen. Sie sind automatisch in der Gruppe ***game.update_sprites.***
visible	Sichtbare Gruppen werden automatisch gezeichnet. Sie sind automatisch in der Gruppe ***game.visible_sprites***.
tcolor	Transparenzfarbe
group	Optionale Gruppe

Automatisch bereitgestellte Eigenschaften

super().*	Die Eigenschaften von GameObject.
fps	Geschwindigkeit der Animation.

Fixe Methoden
Diese Methoden sollten nicht überschrieben werden.

super().*	Alle 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)
animate(value=True)	Startet und 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 Sekunden, die seit dem letzten Aufruf von update() vergangen ist.
draw()	Zeichnen des Elements in self.screen.

Benutzeroberfläche

Label

Ein Label gibt einen Text aus. Der Text kann sehr einfach aktualisiert werden.

Erzeugen

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

game	Das Game wird immer übergeben.
pos	Anzeigeposition
value	Text oder jeder andere Wert, der in einen String umgewandelt werden kann. Wird nur verwendet, wenn weder ***var_name*** noch ***var*** übergeben werden.
fontsize	Grösse des Fonts. Wird nur verwendet, wenn ***font*** nicht übergeben wird.
font	Der Font
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
color	Farbe der Schrift
tcolor	Transparenzfarbe
var_name	Der Name einer registrierten Tracevariablen. Wird nur verwendet, wenn ***var*** nicht übergeben wird.
var	Tracevariable
active	Der Text passt sich nur bei aktiven Objekten an.
visible	Das Label ist sichtbar.

Methoden

`set_value(value)`
Text ändern, wenn keine Tracevariable verwendet wird.

Button
Schaltfläche mit Text.

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

game	Das Game wird immer übergeben.
pos	Anzeigeposition
size	Grösse
text	Der Text, der auf dem Button angezeigt wird.
fontsize	Grösse des Fonts. Wird nur verwendet, wenn ***font*** nicht übergeben wird.
font	Der Font
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
color	Farbe der Schrift
bg_color	Farbe des Buttons
tcolor	Transparenzfarbe
func	Funktion, die aufgerufen werden soll. def func(button): ...
active	Der Text passt sich nur bei aktiven Objekten an.
visible	Das Label ist sichtbar.

Properties

text	Text des Buttons
enabled	
visible	

Checkbox

Erzeugen

```
box = Checkbox(game, pos=(0, 0), size=(10, 10), text="Button", fontsize=50,
               font=None, anchor='tl', color="white", bg_color='darkblue',
               tcolor=(0, 0, 0), func=None, active=True, visible=True,
               checked=False, tag=None)
```

game	Das Game wird immer übergeben.
pos	Anzeigeposition
size	Grösse
text	Der Text, der hinter der Box angezeigt wird.
fontsize	Grösse des Fonts. Wird nur verwendet, wenn ***font*** nicht übergeben wird.
font	Der Font
anchor	Positionsanker, auf den sich ***pos*** bezieht. Standardmässig ist das links oben.
color	Farbe der Schrift
bg_color	Farbe der Box
tcolor	Transparenzfarbe
func	Funktion, die aufgerufen werden soll. def func(box): ...
active	Der Text passt sich nur bei aktiven Objekten an.
visible	Das Label ist sichtbar.
checked	Die Box ist angekreuzt
tag	Eine beliebige Zusatzinformation

Properties

text	Text des Buttons
enabled	
visible	
checked	