Pygame Popup Manager

Release 0.7.0

Grimmys

Aug 14, 2023

CONTENTS

Note: This project is under active development and open to any contribution. Also, don't hesitate to report any typo or unclear statement present in this documentation.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

The easiest way to install this package is to do it with pip:

```
(.venv) $ pip install pygame-popup
```

1.2 Use

pygamepopup.init() function should be call right after pygame.init() function in your code and before any further usage of elements coming from Pygame Popup Manager.

The whole system is working around the *MenuManager* class that is the controller of all the popups in background and in foreground.

An unique instance per scene/window has to be created and the pygame screen on which the popups will appear should be provided, like this:

```
from pygamepopup.menu_manager import MenuManager

screen = pygame.display.set_mode(WINDOW_SIZE)

menu_manager = MenuManager(screen)
```

The *display*, *motion* and *click* methods of this class should be called in the application main loop in order to notify the manager of any pygame event.

To open a new popup menu, you should first create it by instantiate the *InfoBox* class. The components that should be in the menu should be provided.

Next, don't forget to call the *open_menu* method of the menu manager to notify it about the new menu.

For example, the following code will open a popup with a "do-nothing" button and a close button (added by default by the *InfoBox*).

```
from pygamepopup.components import Button, InfoBox

my_custom_menu = InfoBox(
    "Title of the Menu",
    [
        Button(
```

(continues on next page)

(continued from previous page)

```
            title="Hello World!",
            callback=lambda: None
        )
    ]
)

menu_manager.open_menu(my_custom_menu)
```

If you want more code illustrations, go check out the *Examples*.

CHAPTER

TWO

EXAMPLES

2.1 Main menu sample code

```python
import pygame
import pygamepopup
from pygamepopup.components import Button, InfoBox, TextElement
from pygamepopup.menu_manager import MenuManager

WINDOW_WIDTH = 600
WINDOW_HEIGHT = 600


class MainMenuScene:
    def __init__(self, screen: pygame.Surface):
        self.screen = screen
        self.menu_manager = MenuManager(screen)
        self.exit_request = False

        self.create_main_menu_interface()

    def create_main_menu_interface(self):
        main_menu = InfoBox(
            "Main Menu",
            [
                [
                    Button(
                        title="Open other menu",
                        callback=lambda: self.create_other_menu(),
                    )
                ],
                [Button(title="Exit", callback=lambda: self.exit())],
            ],
            has_close_button=False,
        )
        self.menu_manager.open_menu(main_menu)

    def create_other_menu(self):
        other_menu = InfoBox(
            "Smaller menu",
            [
```

(continues on next page)

(continued from previous page)

```
                [
                    TextElement(
                        text="The text content of a menu is automatically splitted in multiple "
                        "part "
                        "to fit in the box. To add a new paragraph, just create another "
                        "TextElement."
                    )
                ]
            ],
            width=300,
        )
        self.menu_manager.open_menu(other_menu)

    def exit(self):
        self.exit_request = True

    def display(self) -> None:
        self.menu_manager.display()

    def motion(self, position: pygame.Vector2) -> None:
        self.menu_manager.motion(position)

    def click(self, button: int, position: pygame.Vector2) -> bool:
        self.menu_manager.click(button, position)
        return self.exit_request


def main() -> None:
    pygame.init()
    pygame.display.set_caption("Main Menu Example")

    pygamepopup.init()

    screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
    main_menu_scene = MainMenuScene(screen)

    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.MOUSEMOTION:
                main_menu_scene.motion(event.pos)
            elif event.type == pygame.MOUSEBUTTONUP:
                if event.button == 1 or event.button == 3:
                    running = not main_menu_scene.click(event.button, event.pos)
        screen.fill(pygame.Color("black"))
        main_menu_scene.display()
        pygame.display.update()
    pygame.quit()
    exit()
```

(continues on next page)

(continued from previous page)

```

if __name__ == "__main__":
    main()
```

2.2 Different in-game menus

```
# pygamepopup simple demo
# Contains 2 menus, independently controllable by keyboard or button
import pygame
import pygamepopup
from pygamepopup.components import Button, InfoBox, TextElement
from pygamepopup.constants import BUTTON_SIZE
from pygamepopup.menu_manager import MenuManager

MAIN_MENU_ID = "main_menu"
CLOSABLE_MENU_ID = "closable_menu"
CUSTOMIZED_MENU_ID = "customized_menu"
SIDE_MENU_ID = "side_menu"

WINDOW_WIDTH = 800
WINDOW_HEIGHT = 600
FPS = 30
clock = pygame.time.Clock()

BALL_RADIUS = 10
x_pos = 250
y_pos = 150
x_velocity = 5
y_velocity = -5

# initialize pygame
pygame.init()
pygame.display.set_caption("Main Menu Example")
screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

# initialize pygamepopup
pygamepopup.init()
menu_manager = MenuManager(screen)

# Define a main menu popup, with 4 buttons and a close button
main_menu = InfoBox(
    "Main Menu",
    [
        [
            Button(
                title="Open side menu",
                callback=lambda: menu_manager.open_menu(side_menu),
                size=(320, BUTTON_SIZE[1]),
            )
```

(continues on next page)

(continued from previous page)

```
        ],
        [
            Button(
                title="Open closable menu",
                callback=lambda: menu_manager.open_menu(closable_menu),
                size=(320, BUTTON_SIZE[1]),
            )
        ],
        [
            Button(
                title="Open customized menu",
                callback=lambda: menu_manager.open_menu(customized_menu),
                size=(320, BUTTON_SIZE[1]),
            )
        ],
        [Button(title="Exit App", callback=lambda: exit(), size=(320, BUTTON_SIZE[1]))],
    ],
    width=420,
    identifier=MAIN_MENU_ID,
)

# Define submenu that can be closed by clicking outside
closable_menu = InfoBox(
    "Closable menu",
    [
        [
            TextElement(
                text="This menu can be close by clicking on the \"Close\" button but␣
↪also by clicking outside of it."
            )
        ]
    ],
    width=350,
    identifier=CLOSABLE_MENU_ID,
)

customized_menu = InfoBox(
    "Customized menu",
    [
        [
            TextElement(
                text="This menu has customized content, such as close button with␣
↪different label"
            )
        ]
    ],
    width=400,
    identifier=CUSTOMIZED_MENU_ID,
    title_color=pygame.Color("red"),
    close_button_text="Shutdown!"
)

```

(continues on next page)

(continued from previous page)

```
# Define a side menu, with relative positioning and close button
side_menu = InfoBox(
    "Smaller menu",
    [
        [
            TextElement(
                text="The text content of a menu is automatically split in multiple "
                "parts "
                "to fit in the box. To add a new paragraph, just create another "
                "TextElement."
            )
        ]
    ],
    element_linked=pygame.Rect(0, WINDOW_HEIGHT // 2, 1, 1),
    width=310,
    identifier=SIDE_MENU_ID,
)


def display_main_screen():
    global x_pos, y_pos, x_velocity, y_velocity
    screen.fill(pygame.Color("tan"))
    font = pygame.font.Font(None, 36)
    text = font.render(
        "Type M to open Main Menu or S to open Side Menu", True, (100, 100, 20)
    )
    screen.blit(text, (screen.get_width() // 2 - text.get_width() // 2, 40))

    # Draw animated Ball
    pygame.draw.circle(screen, pygame.Color("red"), (x_pos, y_pos), BALL_RADIUS, 0)
    x_pos += x_velocity
    y_pos += y_velocity

    if x_pos > WINDOW_WIDTH - BALL_RADIUS or x_pos < BALL_RADIUS:
        x_velocity *= -1
    if y_pos > WINDOW_HEIGHT - BALL_RADIUS or y_pos < BALL_RADIUS:
        y_velocity *= -1


def show_menu(menu):
    # display a menu if it is not already open
    if menu_manager.active_menu is not None:
        if menu_manager.active_menu.identifier == menu.identifier:
            print("Given menu is already opened")
            return
        else:
            menu_manager.close_active_menu()
    menu_manager.open_menu(menu)


def main():
    while True:
```

(continues on next page)

(continued from previous page)

```
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_s:
                    show_menu(side_menu)
                if event.key == pygame.K_m:
                    show_menu(main_menu)
                if event.key == pygame.K_b:
                    print(
                        f"Menus in the background stack: {menu_manager.background_menus}"
                    )
                if event.key == pygame.K_a:
                    print(f"Active menu: {menu_manager.active_menu}")
            elif event.type == pygame.MOUSEMOTION:
                menu_manager.motion(event.pos)  # Highlight buttons upon hover
            elif event.type == pygame.MOUSEBUTTONUP:
                if event.button == 1 or event.button == 3:
                    if menu_manager.active_menu.identifier == CLOSABLE_MENU_ID:
                        if not menu_manager.active_menu.is_position_inside(event.pos):
                            menu_manager.close_active_menu()
                    menu_manager.click(event.button, event.pos)
        display_main_screen()
        menu_manager.display()
        pygame.display.update()
        clock.tick(FPS)


if __name__ == "__main__":
    main()
```

CHAPTER

THREE

MODULE CONTENTS

3.1 Components

3.1.1 BoxElement

class pygamepopup.components.**BoxElement**(**args*, ***kwargs*)

This element acts as a wrapper for a gui element. In fact, it adds a margin to any border of an element to have a cleaner interface.

Keyword Arguments

- **position** (Position) – the position of the box on the screen
- **content** (*Optional[pygame.Surface]*) – a surface that will be wrapped by the box
- **margin** (Margin) – a tuple containing the margins of the box, should be in the form "(top_margin, right_margin, bottom_margin, left_margin), defaults to (0, 0, 0, 0)"

position

the position of the box on the screen

Type
Position

content

the element wrapped in the box

Type
Optional[pygame.Surface]

size

the size of the content following the format "(width, height)"

Type
tuple[int, int]

margin

a dict containing all the values for margins TOP, BOTTOM, LEFT and RIGHT

Type
dict[str, int]

display(*screen*)

Display the content of the box, following the margins that should be added around it.

Keyword Arguments
screen (*pygame.Surface*) – the screen on which the content of the box should be drawn

`get_height()`

Returns
the height of the content more the top and bottom margins

Return type
int

`get_margin_bottom()`

Returns
bottom margin

Return type
int

`get_margin_left()`

Returns
left margin

Return type
int

`get_margin_right()`

Returns
right margin

Return type
int

`get_margin_top()`

Returns
top margin

Return type
int

`get_rect()`

Returns
a pygame rect containing the position of the element and its size

Return type
pygame.Rect

`get_width()`

Returns
the width of the content more the left and right margins

Return type
int

3.1.2 Button

class pygamepopup.components.**Button**(**args*, ***kwargs*)

This class is representing any kind of button that could be seen on an interface.

A button is receptive to user clicks and returns an id corresponding to a method, it may have specific arguments too.

Mouse motion is also handled: the button appearance can change according to current focus.

Keyword Arguments

- **callback** (*Callable*) – the reference to the function that should be call after a click.
- **size** (*tuple[int, int]*) – the size of the button following the format "(width, height)", defaults to BUTTON_SIZE.
- **title** (*str*) – the text that should be displayed at the center of the element.
- **position** (Position) – the position of the element on the screen.
- **background_path** (*str*) – the path to the image corresponding to the sprite of the element.
- **background_hover_path** (*str*) – the path to the image corresponding to the sprite of the element when it has the focus.
- **no_background** (*bool*) – specify whether a background should be present or not, defaults to False.
- **margin** (Margin) – a tuple containing the margins of the box, should be in the form "(top_margin, right_margin, bottom_margin, left_margin)", defaults to (0, 0, 0, 0).
- **disabled** (*bool*) – a boolean indicating if it is not possible to interact with the button, defaults to False.
- **font** (*pygame.font.Font*) – the font that should be used to render the text content.
- **text_color** (*pygame.Color*) – the color of the text content, defautls to WHITE.
- **font_hover** (*pygame.font.Font*) – the font that should be used to render the text content when the mouse is over the button.
- **text_hover_color** (*pygame.Color*) – the color of the text content when the mouse is over the button, defaults to WHITE.
- **complementary_text_lines** (*str*) – the other text lines that should be displayed in addition of the title.

callback

the reference to the function that should be call after a click.

Type
Callable

sprite

the pygame Surface corresponding to the sprite of the element.

Type
pygame.Surface

sprite_hover

the pygame Surface corresponding to the sprite of the element when it has the focus.

Type
pygame.Surface

`action_triggered`()

Method that should be called after a click.

Returns
the callback that should be executed, a callback doing nothing would be returned if the button is disabled.

Return type
Callable

`render_sprite`(*background_path*, *rendered_text_lines*)

Compute the rendering of the button with the given background and text lines. If no background is provided, render the text on an empty surface.

Returns
the generated surface.

Return type
pygame.Surface

Keyword Arguments

- **background_path** (*str*) – the path to the image corresponding to the sprite of the button.
- **rendered_text_lines** (*Sequence[pygame.Surface]*) – the sequence of text lines in order that should be clipped on the surface.

`static render_text_lines`(*text_lines*, *text_color*, *font*)

Compute the rendering of the given text.

Returns
the rendered text lines.

Return type
Sequence[pygame.Surface]

Keyword Arguments

- **text_lines** (*Sequence[str]*) – the sequence in order of text lines to be rendered.
- **text_color** (*pygame.Color*) – the color of the text.
- **font** (*pygame.font.Font*) – the font that should be used to render the text.

`set_hover`(*is_mouse_hover*)

Change the current sprite between sprite or sprite_hover depending on whether the mouse is over the element or not.

Keyword Arguments
is_mouse_hover (*bool*) – a boolean value indicating if the mouse is over the element or not

3.1.3 DynamicButton

class pygamepopup.components.**DynamicButton**(**args*, ***kwargs*)

This class is representing a special button with an inner value changing after each click.

A DynamicButton has a sequence of values given at initialization, and a initial value.

The sequence will be iterated to determine the next inner value after a click.

This fluctuating value is the one that will be send has the first argument of the method called on click, and a different label will be displayed on the button for each different value of the sequence.

Keyword Arguments

- **callback** (*Callable*) – the reference to the function that should be call after a click.
- **values** (*Sequence[any]*) – the sequence of values that will be iterated to determine the next inner value.
- **current_value_index** (*int*) – the index of the initial value of the button.
- **base_title** (*str*) – the common prefix of all the different labels (it could be the name of the dynamic button in a way).
- **size** (*str*) – the size of the button following the format "(width, height)", defaults to BUTTON_SIZE.
- **position** (Position) – the position of the element on the screen.
- **background_path** (*str*) – the path to the image corresponding to the sprite of the element.
- **background_hover_path** (*str*) – the path to the image corresponding to the sprite of the element when it has the focus.
- **no_background** (*bool*) – specify whether a background should be present or not, defaults to False.
- **margin** (Margin) – a tuple containing the margins of the box, should be in the form "(top_margin, right_margin, bottom_margin, left_margin)", defaults to (0, 0, 0, 0).
- **disabled** (*bool*) – a boolean indicating if it is not possible to interact with the button, defaults to False.

values

the sequence of values that will be iterated to determine the next inner value.

Type
Sequence[any]

current_value_index

the index of the current value of the button.

Type
int

base_title

the common prefix of all the different labels (it could be the name of the dynamic button in a way).

Type
str

`action_triggered`()

Method that should be called after a click.

Change the current value of the button to the next one in the sequence of values. If the end of the sequence is reach, the iteration restarts at the first value.

Returns
a lambda containing the function that should be called with the current value of the dynamic button as an argument.

Return type
Callable

3.1.4 ImageButton

`class` pygamepopup.components.`ImageButton`(**args*, ***kwargs*)

This component is a Button having an image displayed at the left of its content and that is considered as part of the button.

The image is displayed in a dedicated square frame.

Keyword Arguments

- **callback** (*Callable*) – the reference to the function that should be call after a click.
- **size** (*tuple[int, int]*) – the size of the button following the format "(width, height)", defaults to IMAGE_BUTTON_SIZE.
- **title** (*str*) – the text that should be displayed at the center of the element.
- **position** (Position) – the position of the element on the screen.
- **background_path** (*str*) – the path to the image corresponding to the sprite of the element.
- **frame_background_path** (*str*) – the path to the background for the frame.
- **frame_background_hover_path** (*str*) – the path to the background for the frame when the mouse is over.
- **margin** (Margin) – a tuple containing the margins of the box, should be in the form "(top_margin, right_margin, bottom_margin, left_margin)", defaults to (0, 0, 0, 0).
- **disabled** (*bool*) – a boolean indicating if it is not possible to interact with the button, defaults to False.
- **font** (*pygame.font.Font*) – the font that should be used to render the text content.
- **text_color** (*pygame.Color*) – the color of the text content, defautls to WHITE.
- **font_hover** (*pygame.font.Font*) – the font that should be used to render the text content when the mouse is over the button.
- **text_hover_color** (*pygame.Color*) – the color of the text content when the mouse is over the button, defaults to MIDNIGHT_BLUE.
- **complementary_text_lines** (*str*) – the other text lines that should be displayed in addition of the title.
- **image_path** (*str*) – the relative path to the image that should be displayed on the left (inside of the frame).

render_sprite(*background_path*, *rendered_text_lines*)

Compute the rendering of the image button with the given background and text lines.

Render the text lines next to the image frame.

Returns
the generated surface.

Return type
pygame.Surface

Keyword Arguments

- **background_path** (*str*) – the path to the image corresponding to the sprite of the button
- **rendered_text_lines** (*Sequence[pygame.Surface]*) – the sequence of text lines in order that should be clipped on the surface

3.1.5 InfoBox

class pygamepopup.components.**InfoBox**(*title*, *element_grid*, *width=400*, *element_linked=None*, *position=None*, *has_close_button=True*, *title_color=(255, 255, 255, 255)*, *background_path=None*, *close_button_text=None*, *close_button_background_path=None*, *close_button_background_hover_path=None*, *visible_on_background=True*, *has_vertical_separator=False*, *identifier=''*)

This class is defining any kind of popup that can be found in the app.

It can be used to represent the interface of a menu, or a simple text message. Some elements can be buttons, that will react to user clicks (see the button component for more information).

Keyword Arguments

- **title** (*str*) – the title of the infoBox
- **element_grid** (*list[list[BoxElement]]*) – a grid containing the components that should be rendered by the infoBox
- **width** (*int*) – the width of the infoBox, defaults to DEFAULT_POPUP_WIDTH
- **element_linked** (*pygame.Rect*) – the pygame Rect of the element linked to this infoBox, the infoBox will be displayed beside the element if provided
- **position** (*Position*) – the static position of the infoBox, if not provided position would be computed basing on
- **screen** (*element linked or*) –
- **has_close_button** (*bool*) – whether a close button should be added at the bottom or not, defaults to True
- **title_color** (*pygame.Color*) – the color of the title
- **background_path** (*str*) – the path corresponding to the image that should be the sprite of the infoBox
- **close_button_text** (*str*) – the text that will be shown on close button
- **close_button_background_path** (*str*) – the path to the image corresponding to the sprite of the close button if there should be one

- **close_button_background_hover_path** (*str*) – the path to the image corresponding to the sprite of the close button when it is hovered if there should be one
- **visible_on_background** (*bool*) – whether the popup is visible on background or not, defaults to True
- **has_vertical_separator** (*bool*) – whether there should be a line splitting the infoBox in two at middle width or not, defaults to False
- **identifier** (*str*) – a string permitting to identify the menu among others if needed

title

the title of the infoBox

Type
str

element_linked

the pygame Rect of the element linked to this infoBox if there is one

Type
pygame.Rect

has_close_button

whether the infoBox has a close button or not

Type
bool

title_color

the color of the title

Type
pygame.Color

element_grid

the grid containing the components that should be rendered by the infoBox

Type
list[list[*BoxElement*]

buttons

the sequence of buttons of the infoBox, including the close button if present

Type
Sequence[*Button*]

sprite

the pygame Surface corresponding to the sprite of the infoBox

Type
pygame.Surface

visible_on_background

whether the popup is visible on background or not

Type
bool

identifier

a string permitting to identify the menu among others if needed

Type
str

`click`(*position*)

Handle the triggering of a click event.

Returns

the data corresponding to the action that should be done if the click was done
on a button, else None.

Return type
Optional[Callable]

Keyword Arguments
position (Position) – the position of the mouse

`determine_elements_position`()

Compute the position of each element and update it if needed.

`determine_position`(*screen*)

Compute the position of the infoBox to be beside the linked element.

If no element is linked to the infoBox, the position will be determined at display time according to the screen.

Returns
the computed position.

Return type
Optional[Position]

Keyword Arguments
screen (*pygame.Surface*) – The screen on which the infoBox is rendered.

`display`(*screen*)

Display the infoBox and all its elements.

Keyword Arguments
screen (*pygame.Surface*) – the screen on which the displaying should be done

`find_buttons`()

Search in all elements for buttons.

Returns
the sequence of buttons.

Return type
Sequence[*Button*]

`init_elements`()

Initialize the graphical elements associated to the formal data that the infoBox should represent.

Returns
the elements in a 2D structure corresponding to the relative position of each element.

`init_render`(*screen*, *close_button_callback=None*)

Initialize the rendering of the popup.

Compute it size and its position according to the given screen. Determine the position of each component.

Keyword Arguments

- **screen** (*pygame.Surface*) – the screen on which the popup is
- **close_button_callback** (*Callable*) – the callback that should be executed when clicking on the close button if there is any

is_position_inside(*position*)

Check if given position is inside or outside the InfoBox.

Returns
whether the position is inside the borders or not

Return type
bool

Keyword Arguments
position (Position) – the position to be checked

motion(*position*)

Handle the triggering of a motion event. Test if the mouse entered a button or left one.

Keyword Arguments
position (Position) – the position of the mouse

3.1.6 TextElement

class pygamepopup.components.**TextElement**(**args*, ***kwargs*)

This class is representing a paragraph of text, displayed on a popup and horizontally centered according to its position.

Keyword Arguments

- **text** (*str*) – the text that should be rendered.
- **position** (Position) – the position of the text on the screen.
- **font** (*pygame.font.Font*) – the font that should be used to render the text.
- **margin** (Margin) – a tuple containing the margins of the box, should be in the form "(top_margin, right_margin, bottom_margin, left_margin)", defaults to (0, 0, 0, 0).
- **text_color** (*pygame.Color*) – the color of the rendered text, defaults to WHITE.

3.2 Configuration

Defines utility functions to configure the default values used across the library.

pygamepopup.configuration.**set_button_background**(*button_background_path*, *button_hovered_background_path*)

Set the default backgrounds for buttons.

Keyword Arguments

- **button_background_path** (*str*) – the path to the background sprite to be set.
- **button_hovered_background_path** (*str*) – the path to the background sprite when hovering to be set.

pygamepopup.configuration.**set_button_title_font**(*font*)

Set the default font for the title of the buttons.

Keyword Arguments
font (*pygame.font.Font*) – the loaded font to be set.

pygamepopup.configuration.**set_close_button_text**(*text*)

Set the default text for the close button of InfoBox.

Keyword Arguments
text (*str*) – the default text to be set.

pygamepopup.configuration.**set_dynamic_button_background**(*button_background_path*, *button_hovered_background_path*)

Set the default backgrounds for dynamic buttons.

Keyword Arguments

- **button_background_path** (*str*) – the path to the background sprite to be set.
- **button_hovered_background_path** (*str*) – the path to the background sprite when hovering to be set.

pygamepopup.configuration.**set_dynamic_button_title_font**(*font*)

Set the default font for the title of the dynamic buttons.

Keyword Arguments
font (*pygame.font.Font*) – the loaded font to be set.

pygamepopup.configuration.**set_info_box_background**(*info_box_background_path*)

Set the default background for infoboxes.

Keyword Arguments
info_box_background_path (*str*) – the path to the background sprite to be set.

pygamepopup.configuration.**set_info_box_title_font**(*font*)

Set the default font for the title of InfoBox.

Keyword Arguments
font (*pygame.font.Font*) – the loaded font to be set.

pygamepopup.configuration.**set_text_element_font**(*font*)

Set the default font for the text of the text elements.

Keyword Arguments
font (*pygame.font.Font*) – the loaded font to be set.

3.3 Initialization

pygamepopup.**init**()

Initialize the package modules.

Must be called before all other calls and after the initialization of pygame.

3.4 MenuManager

class pygamepopup.menu_manager.**MenuManager**(*screen*)

This class represents a manager for the interfaces of a screen.

It displays all the menus currently registered in the manager.

Handle the opening of a new menu and the closing of the active one. Handle the triggering of user motion events and user click events on the active menu.

Keyword Arguments
screen (*pygame.Surface*) – the screen on which the menus should be displayed and on which the user events should be handled

screen

the screen on which the menus should be displayed and on which the user events should be handled

Type
pygame.Surface

active_menu

the current menu in the foreground, the only one that will react to user events

Type
Optional[*InfoBox*]

background_menus

the ordered sequence of menus that are in the background

Type
list[*InfoBox*]

clear_menus()

Close all the menus (in foreground and in background)

click(*button*, *position*)

Handle the triggering of a click event. Delegate this event to the active menu if there is any and if it's a left click.

Keyword Arguments

- **button** (*int*) – a value representing which mouse button has been pressed (1 for left button, 2 for middle button, 3 for right button)
- **position** (*Position*) – the position of the mouse

close_active_menu()

Close the active menu by 'destroying' it.

Take the next menu in the background to move it to foreground if there is any.

close_given_menu(*menu_identifier*, *all_occurrences=False*)

Close menu corresponding to the given identifier.

By default, only first occurrence is closed if many menus in the manager have the given identifier.

Keyword Arguments

- **menu_identifier** (*str*) – the identifier of the menu to be closed
- **all_occurrences** (*bool*) – whether all found occurrences should be closed or only the first one, defaults to False

Returns
whether at least one menu has been closed or not

Return type
bool

`display()`

Display all the visible menus in the background in order first, then display the active menu

`motion(position)`

Handle the triggering of a motion event. Delegate this event to the active menu if there is any.

Keyword Arguments
position (Position) – the position of the mouse

`open_menu(menu)`

Open the given menu. Initialize the rendering of the menu, and set it as the new active menu.

Previous active menu is sent to the background.

Keyword Arguments
menu (InfoBox) – the popup that should be open

`reduce_active_menu()`

Move the active menu to the background.

`replace_given_menu(menu_identifier, new_menu, all_occurrences=False)`

Replace a menu by a new one according to its identifier.

By default, only first occurrence is replaced if many menus in the manager have the given identifier.

Keyword Arguments

- **menu_identifier** (*str*) – the identifier of the menu to be replaced
- **new_menu** (InfoBox) – the menu that should replace the given one
- **all_occurrences** (*bool*) – whether all found occurrences should be replaced or only the first one

Returns
whether the replacement has succeeded or not

Return type
bool

3.5 Types

Defines typing for structures having specific meaning.

`pygamepopup.types.Margin`

Alias for tuple[int, int, int, int]

`pygamepopup.types.Position`

Alias for pygame.Vector2. A tuple[int, int] can alternatively be given in place of it.

alias of Union[Vector2, tuple[int, int]]

CHAPTER

FOUR

LINKS & CONTACT

Link to the package in pypi: https://pypi.org/project/pygame-popup

Link to the source code: https://github.com/Grimmys/pygame_popup_manager
Feel free to create **issues** or suggest **pull requests** there.

Even if it's better to fill an issue so everybody can see what is going on, you can alternatively contact the main maintainer of this library by e-mail.

PYTHON MODULE INDEX

p

INDEX

H

I

M

O

P

R

S

T

V