2. Learn Basic Game Designing Techniques with pygame

Pygame Introduction

- Pygame is a cross-platform set of Python modules which is used to create video games.
- It consists of computer graphics and sound libraries designed to be used with the Python programming language.
- Pygame was officially written by **Pete Shinners** to replace PySDL.
- Pygame is suitable to create client-side applications that can be potentially wrapped in a standalone executable.

Pygame Installation

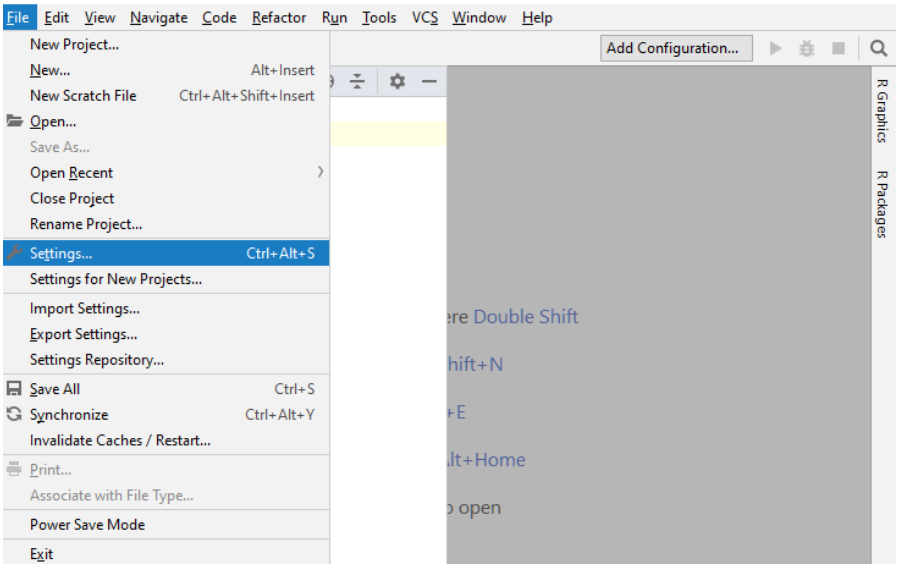
- Before installing Pygame, Python should be installed in the system, and it is good to have 3.6.1 or above version because it is much friendlier to beginners, and additionally runs faster.
- There are mainly two ways to install Pygame, which are given below:

1. Installing through pip: The good way to install Pygame is with the pip tool (which is what python uses to install packages). The command is the following:

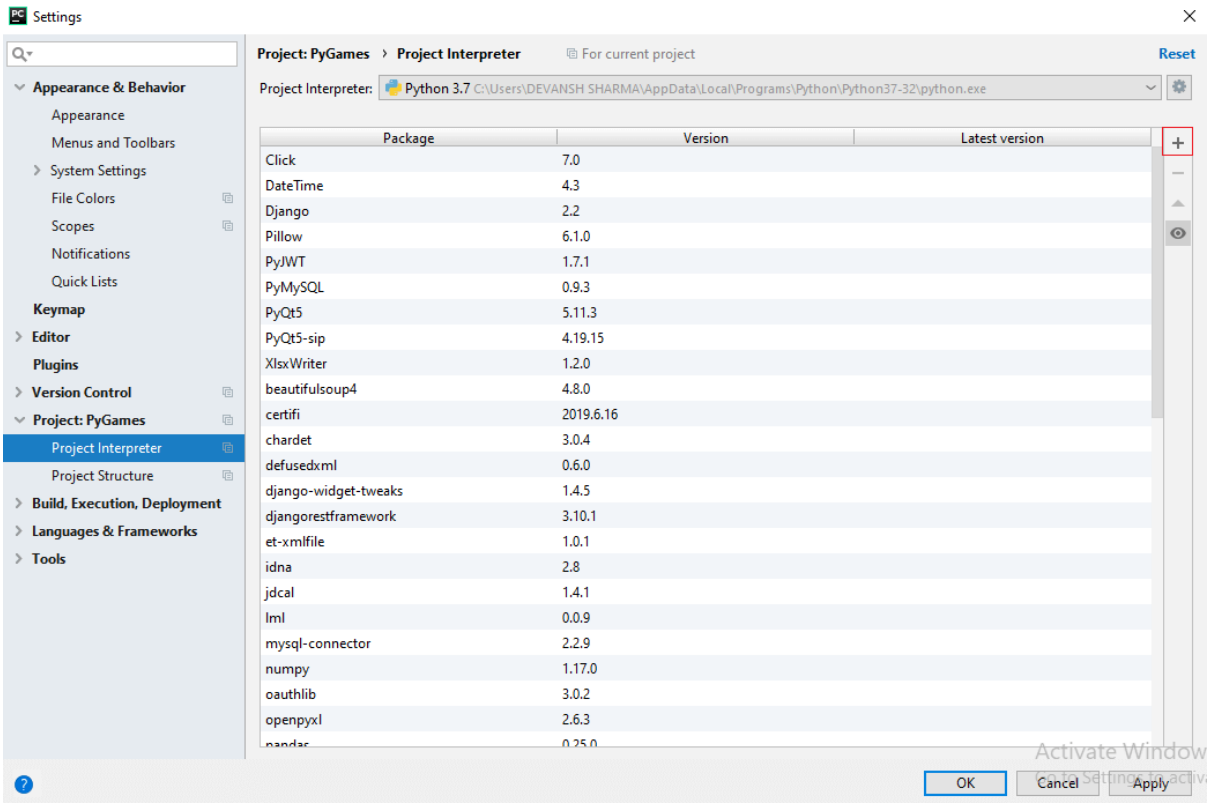
```
py -m pip install -U pygame --user
```

2. Installing through an IDE: The second way is to install it through an IDE and here we are using Pycharm IDE. Installation of pygame in the pycharm is straightforward. We can install it by running the above command in the terminal or use the following steps:

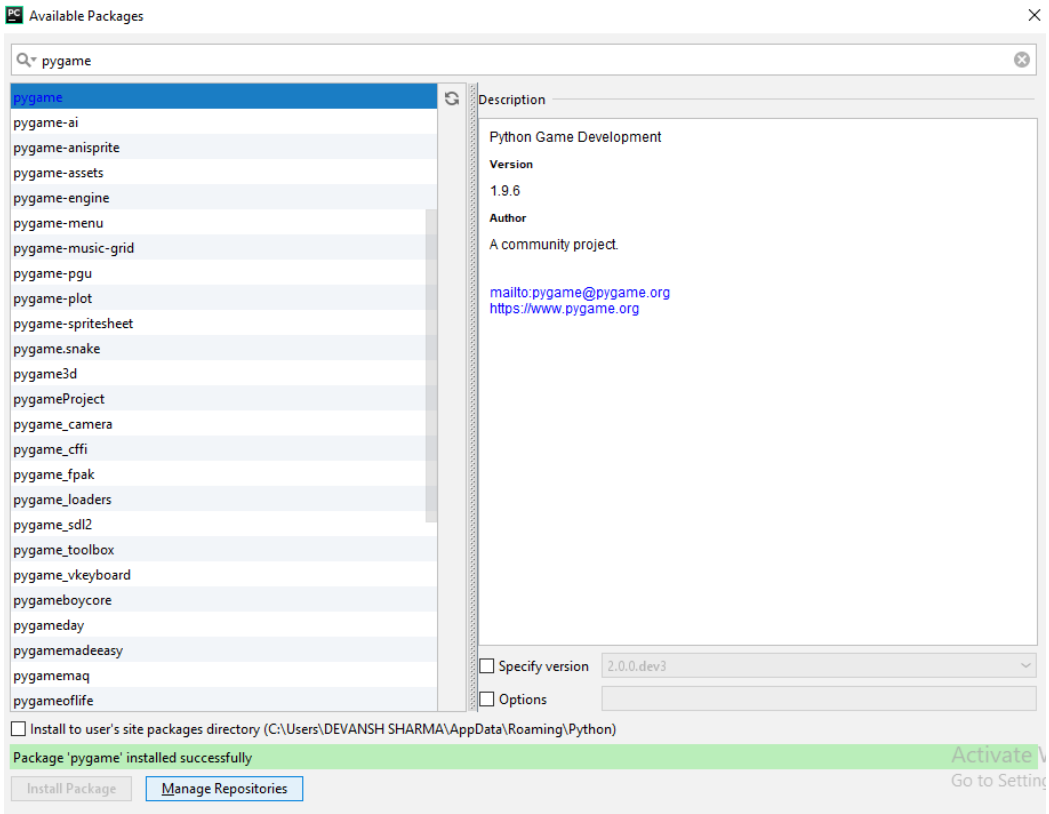
- Open the **File** tab and click on the **Settings** option.



- Select the **Project Interpreter** and click on the + icon.



- It will display the search box. Search the pygame and click on the **install package** button.



Note: ***You can use Google Colab or any other jupyter notebook for performing practical.***

Verifying whether pygame is correctly installed or not

To check whether the pygame is properly installed or not, in the IDLE interpreter, type the following command and press Enter:

import pygame

If the command runs successfully without throwing any errors, it means we have successfully installed Pygame and found the correct version of IDLE to use for pygame programming.

Simple pygame Example

```
1. import pygame
2.
3. pygame.init()
4. screen = pygame.display.set_mode((400,500))
5. done = False
6.
7. while not done:
8.     for event in pygame.event.get():
9.         if event.type == pygame.QUIT:
10.            done = True
11.    pygame.display.flip()
```

This provides access to the pygame framework and imports all functions of pygame.

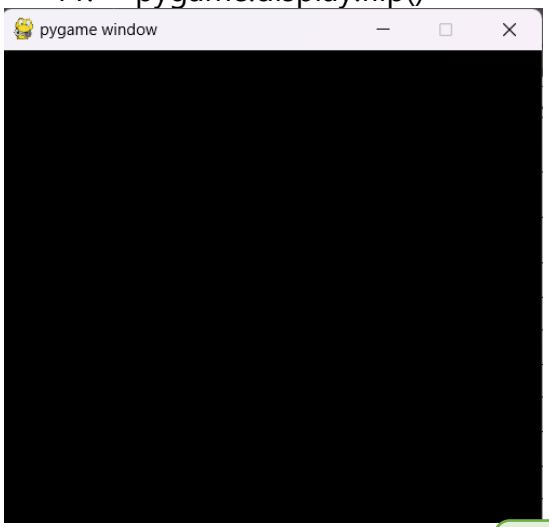
This is used to initialize all the required module of the pygame.

this is used to display a window of the desired size. The return value is a Surface object which is the object where we will perform graphical operations.

This is used to empty the event queue. If we do not call this, the window messages will start to pile up and, the game will become unresponsive in the opinion of the operating system.

This is used to terminate the event when we click on the close button at the corner of the window.

Pygame is double-buffered, so this shifts the buffers. It is essential to call this function in order to make any updates that you make on the game screen to make visible.



Output:

Pygame Surface

- The pygame Surface is used to display any image.
- The Surface has a pre-defined resolution and pixel format.
- The Surface color is by default black.
- Its size is defined by passing the **size** argument.
- Surfaces can have the number of extra attributes like **alpha planes, color keys, source rectangle clipping, etc**.
- The blit routines will attempt to use hardware acceleration when possible; otherwise, they will use highly enhanced software blitting methods.

Pygame Clock

- Times are represented in millisecond (1/1000 seconds) in pygame.
- Pygame clock is used to track the time.
- The time is essential to create motion, play a sound, or, react to any event.
- In general, we don't count time in seconds. We count it in milliseconds.
- The clock also provides various functions to help in controlling the game's frame rate. The few functions are the following:

tick()

This function is used to update the clock. The syntax is the following:

tick(framerate=0)

- This method should be called once per frame.
- It will calculate how many milliseconds have passed since the previous call.
- The **framerate** argument is optional to pass in the function, and if it is passed as an argument then the function will delay to keep the game running slower than the given ticks per second.

tick_busy_loop()

- The tick_busy_loop() is same as the tick().
- By calling the **Clock.tick_busy_loop(20)** once per frame, the program will never run at more than 20 frames per second.
- The syntax is the following:

tick_busy_loop()

get_time()

- The get_time() is used to get the previous tick.

- The number of a millisecond that isdra passed between the last two calls in Clock.tick().

get_time()

Pygame Blit

- The pygame blit is the process to render the game object onto the surface, and this process is called **blitting**.
- When we create the game object, we need to render it.
- If we don't render the game objects and run the program, then it will give the black window as an output.
- Blitting is one of the slowest operations in any game so, we need to be careful to not to blit much onto the screen in every frame.
- The primary function used in blitting is blit(), which is:

blit()

```
blit(source,dest,area=None,special_flags=0)
```

- This function is used to draw one image into another.
- The draw can be placed with the dest argument.
- The dest argument can either be a pair of coordinates representing the upper left corner of the source.

Pygame Adding Image

- To add an image on the window, first, we need to instantiate a blank surface by calling the Surface constructor with a width and height tuple.

```
surface = pygame.Surface((100,100))
```

- The above line creates a blank 24-bit RGB image that's 100*100 pixels with the default black color.
- For the transparent initialization of Surface, pass the SRCALPHA argument.

```
surface = pygame.Surface((100,100), pygame.SRCALPHA)
```

Program to display an Image

Consider the following example to display image on the surface:

```
1. import pygame
2. pygame.init()
3. white = (255, 255, 255)
```

```
# assigning values to height and width variable
height = 400
width = 400
# creating the display surface object
# of specific dimension..e(X, Y).
display_surface = pygame.display.set_mode((height, width))

# set the pygame window name
pygame.display.set_caption('Image')

# creating a surface object, image is drawn on it.
image = pygame.image.load('C:\Users\ \Desktop\download.png')

# infinite loop
while True:
    display_surface.fill(white)
    display_surface.blit(image, (0, 0))

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            # quit the program.
            quit()
        # Draws the surface object to the screen.
        pygame.display.update()
```

You need to give path of an image

Pygame Rect

- Rect is used to draw a rectangle in Pygame.
- Pygame uses Rect objects to store and manipulate rectangular areas.
- A Rect can be formed from a combination of left, top, width, and height values.
- It can also be created from Python objects that are already a Rect or have an attribute named "rect".
- The **rect()** function is used to perform changes in the position or size of a rectangle.
- It returns the new copy of the Rect with the affected changes.

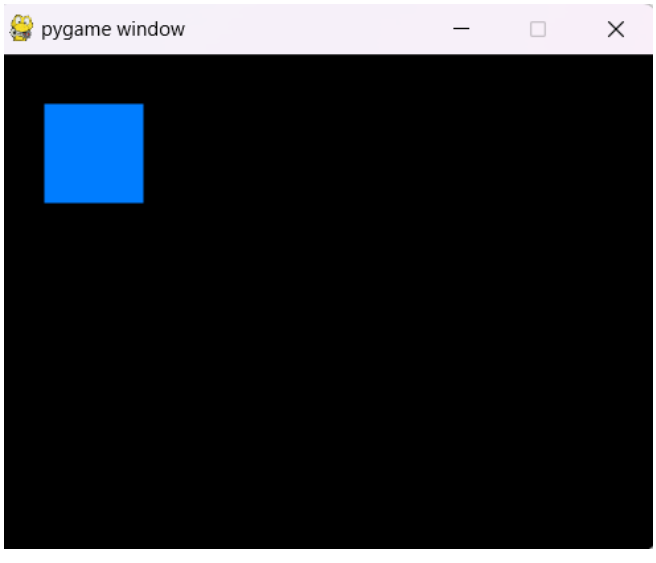
- No modification happens in the original rectangle.
- The Rect object has various virtual attributes which can be used to move and align the Rect:

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

- The dimension of the rectangle can be changed by assigning the size, width, or height.
- All other assignment moves the rectangle without resizing it.
- If the width or height is a non-zero value of Rect, then it will return True for a non-zero test.
- Some methods return a Rect with 0 sizes to represent an invalid rectangle.

Program to create a rectangle

```
1. import pygame
2.
3. pygame.init()
4. screen = pygame.display.set_mode((400, 300))
5. done = False
6.
7. while not done:
8.     for event in pygame.event.get():
9.         if event.type == pygame.QUIT:
10.            done = True
11.    pygame.draw.rect(screen, (0, 125, 255), pygame.Rect(30, 30, 60, 60))
12.
13.    pygame.display.flip()
```



Pygame Keydown

Pygame KEYDOWN and KEYUP detect the event if a key is physically pressed and released. **KEYDOWN** detects the key press and, **KEYUP** detects the key release. Both events (Key press and Key release) have two attributes which are the following:

- **key:** Key is an integer id which represents every key on the keyword.
- **mod:** This is a bitmask of all the modifier keys that were in the pressed state when the event occurred.

Consider the following example of the key press and key release.

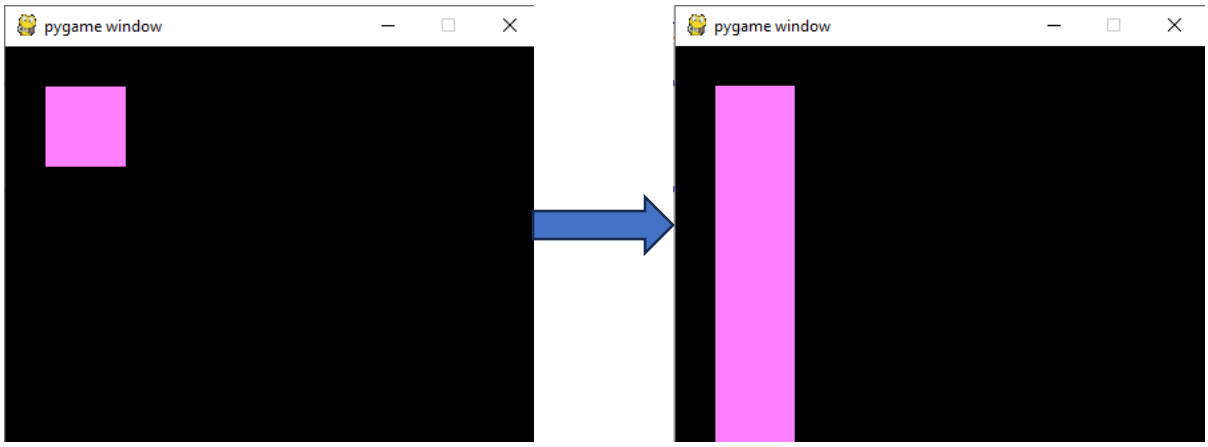
```
1.  import pygame
2.  pygame.init()
3.  # sets the window title
4.  pygame.display.set_caption(u'Keyboard events')
5.  # sets the window size
6.  pygame.display.set_mode((400, 400))
7.
8.  while True:
9.      # gets a single event from the event queue
10.     event = pygame.event.wait()
11.     # if the 'close' button of the window is pressed
12.     if event.type == pygame.QUIT:
13.         # stops the application
14.         break
15.     # Detects the 'KEYDOWN' and 'KEYUP' events
16.     if event.type in (pygame.KEYDOWN, pygame.KEYUP):
17.         # gets the key name
18.         key_name = pygame.key.name(event.key)
19.         # converts to uppercase the key name
20.         key_name = key_name.upper()
21.         # if any key is pressed
22.         if event.type == pygame.KEYDOWN:
23.             # prints on the console the key pressed
24.             print(u'"{}" key pressed'.format(key_name))
25.         # if any key is released
26.         elif event.type == pygame.KEYUP:
```

```
27.         # prints on the console the released key
28.         print(u'"{}" key released'.format(key_name))
```

Output :

In the above code, the rectangle will be displayed on the pygame window.

When we press the Down key, the rectangle is reshaped in the downwards.



Pygame Draw

- Pygame provides geometry functions to draw simple shapes to the surface.
- These functions will work for rendering to any format to surfaces.
- Most of the functions accept a width argument to signify the size of the thickness around the edge of the shape.
- If the width is passed 0, then the shape will be solid(filled).
- All the drawing function takes the color argument that can be one of the following formats:
    - A pygame.Color objects
    - An (RGB) triplet(tuple/list)
    - An (RGBA) quadruplet(tuple/list)
    - An integer value that has been mapped to the surface's pixel format

Draw a rectangle

The following functions are used to draw a rectangle on the given surface.

1. pygame.draw.rect(surface, color, rect)
2. pygame.draw.rect(surface, color, rect, width=0)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **rect(Rect)-** Draw rectangle, position, and dimensions.
- **width(int)-** This is optional to use the line thickness or to indicate that the rectangle is filled.

1. **if** width == 0, (**default**) fill the rectangle
2. **if** width > 0, used **for** line thickness
3. **if** width < 0, nothing will be drawn

Draw a polygon

The following functions are used to draw a polygon on the given surface.

- pygame.draw.polygon(surface,color,points)
- pygame.draw.polygon(surface, color, points, width=0)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **points(tuple(coordinate) or list(coordinate)):** A sequence of 3 or more (x,y) coordinates that make up the vertices of the polygon. Each coordinate in the sequence must be tuple/list.

Draw an ellipse

The following functions are used to draw an ellipse on the given surface.

1. pygame.draw.ellipse(surface, color, rect)
2. pygame.draw.ellipse(surface, color, rect, width=0)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **rect(Rect)-** Draw rectangle, position, and dimensions.

Draw a straight line

This method is used to draw a straight line on the given surface. There are no endcaps.

1. pygame.draw.line(surface,color,start_pos,end_pos,width)
2. pygame.draw.line(surface,color,start_pos,end_pos,width=1)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **start_pos-** start position of the line(x,y)
- **end_pos-** End position of the line

Draw a Circle

Below are the functions, which are used to draw a circle on the given surface.

- circle(surface, color, center, radius)
- circle(surface, color, center, radius, width=0)

Parameters:

- **surface -** Screen to draw on.
- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **center -** The center point of the circle as a sequence of two int/float, e.g. (x,y)
- **radius(int or float)-** radius of the circle, measured from the center parameter, if the radius is zero, then it will only draw the center pixel.

Draw an elliptical arc

Below functions are used to draw an elliptical arc on the given surface.

1. ? arc(surface, color, rect, start_angle, stop_angle)
2. ? arc(surface, color, rect, start_angle, stop_angle, width=1)

Parameters:

- **surface -** Screen to draw on.

- **color-** This argument is used to color the given shape. The alpha value is optional if we are using a tuple.
- **rect(Rect)-** Draw rectangle, position, and dimensions.
- **start_angle-** Start angle of the arc in radians.
- **stop_angle-** Stop angle of the arc in radians.

There are three conditions for start_angle and stop_angle parameter:

a. If start_angle < stop_angle then the arc will be drawn in a counter-clock direction from the start_angle to end_angle.

b. If start_angle>stop_angle then tau(tau=2*pi) will be added to the stop angle.

c. If start_angle==stop_angle, nothing will be drawn.

Pygame Text and Font

- Pygame also provides facilities to render the font and text.
- We can load fonts from the system by using the **pygame.font.SysFont()** function.
- Pygame comes with the built-in default font which can be accessed by passing the font name or None.
- There are many functions to help to work with the font.
- The font objects are created with **pygame.font.Font()**.
- The actual font objects do most of the works done with fonts.
- Font objects are generally used to render the text into new Surface objects.
- Few important font functions are the following:

render()

- This function is used to draw text on a new Surface.
- Pygame has no facility to draw text on the existing Surface.
- This creates a new Surface with the specified text render on it.
- The syntax is the following:

render(text, antialias, color, background=None)

size()

- This function is used to determine the number of space or positioning needed to render text.

- It can also be used for word-wrapping and other layout effects.
- The syntax is the following:

size(bool)

set_bold()

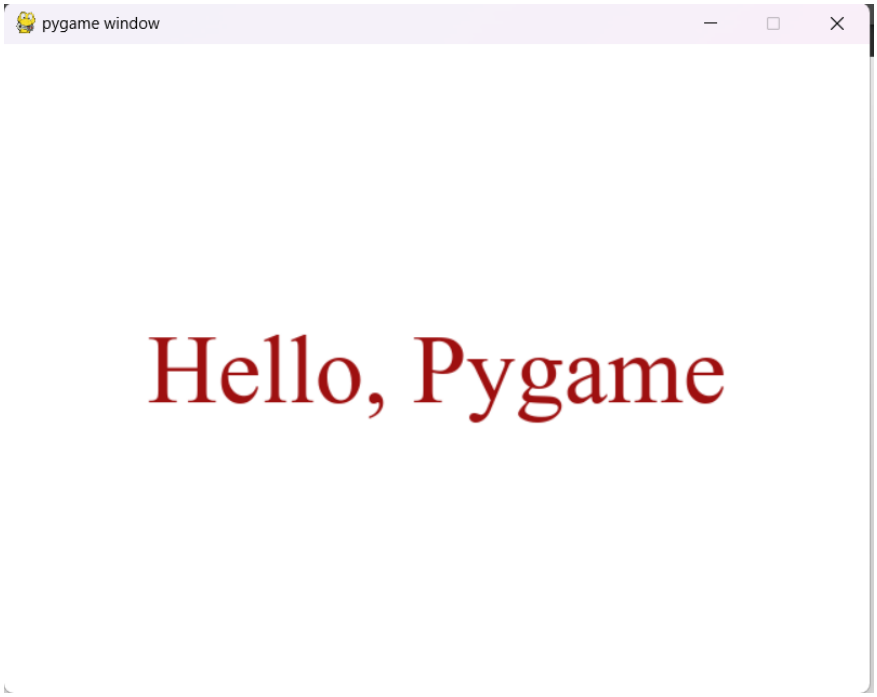
This function is used for bold rending of text. The syntax is following:

set_bold(bool)

```
import pygame
pygame.init()
screen = pygame.display.set_mode((640, 480))
done = False

#load the fonts
font = pygame.font.SysFont("Times new Roman", 72)
# Render the text in new surface
text = font.render("Hello, Pygame", True, (158, 16, 16))
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        if event.type == pygame.KEYDOWN and event.key == pygame.K_ESCAPE
    :
            done = True
    screen.fill((255, 255, 255))
    #We will discuss blit() in the next topic
    screen.blit(text,(320 - text.get_width() // 2, 240 - text.get_height() // 2))

    pygame.display.flip()
```



Pygame Sprite and Collision detection

A pygame sprite is a two-dimensional image that is part of the large graphical scene. Usually, a sprite will be some object in the scene.

One of the most advantages of working with sprites is the ability to work with them in groups. We can easily move and draw all the sprites with the one command if they are in the group.

The Sprite module contains the various simple classes to be used within the games. It is optional to use Sprite classes and different group classes when using pygame.

Pygame provides sprites and sprite groups that help for collision detection. Collision detection is the process when two objects on the screen collide each other. For example, if a player is hit by the enemy's bullet, then it may lose a life or, the program need to know when the player touches a coin so that they automatically picked up.