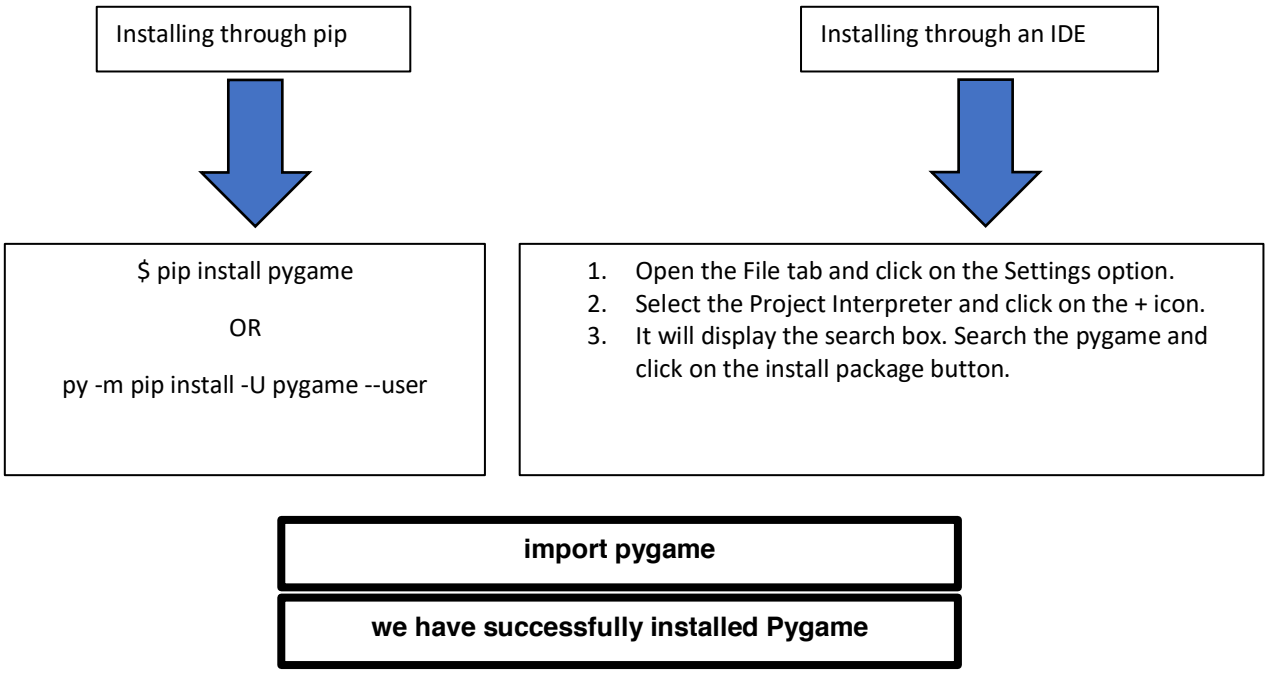
Pygame

Python is the most popular programming language or nothing wrong to say that it is the next-generation programming language. In every emerging field in computer science, Python makes its presence actively.

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Pygame was originally written by Pete Shinners to replace PySDL after its development stalled. It has been a community project since 2000 and is released under the open source free software GNU Lesser General Public License.

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:

1. **Installing through pip:** The good way to install Pygame is with the pip tool (which is what python uses to install packages).
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: 1. Open the File tab and click on the Settings option. 2. Select the Project Interpreter and click on the + icon. 3. It will display the search box. Search the pygame and click on the install package button.

To check whether the pygame is properly installed or not, in the IDLE interpreter, type the **import pygame** command and press Enter.

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

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.

Modules in the Pygame Package

Module Name	Purpose
pygame.cdrom	Accesses and controls CD drives
pygame.cursors	Loads cursor images
pygame.display	Accesses the display
pygame.draw	Draws shapes, lines, and points
pygame.event	Manages external events
pygame.font	Uses system fonts
pygame.image	Loads and saves an image
pygame.joystick	Uses joysticks and similar devices
pygame.key	Reads key presses from the keyboard
pygame.mixer	Loads and plays sounds
pygame.mouse	Manages the mouse
pygame.movie	Plays movie files
pygame.music	Works with music and streaming audio
pygame.overlay	Accesses advanced video overlays
pygame	Contains high-level Pygame functions
pygame.rect	Manages rectangular areas
pygame.sndarray	Manipulates sound data
pygame.sprite	Manages moving images
pygame.surface	Manages images and the screen
pygame.surfarray	Manipulates image pixel data
pygame.time	Manages timing and frame rate
pygame.transform	Resizes and moves images

*For complete documentation on the Pygame modules, see www.pygame.org/docs/.

The Pygame package contains a number of modules that can be used independently. There is a module for each of the devices that you might use in a game, and many others to make game creation a breeze.

You access these modules through the ***pygame*** namespace; for instance, ***pygame.display*** refers to the display module.

Some of the modules you will use in every game. You will always have some sort of display, so the display module is essential, and you will definitely need some kind of input, whether it is keyboard, joystick, or mouse. Other modules are less commonly used, but in combination they give you one of the most powerful game creation tools around.

Simple pygame Example

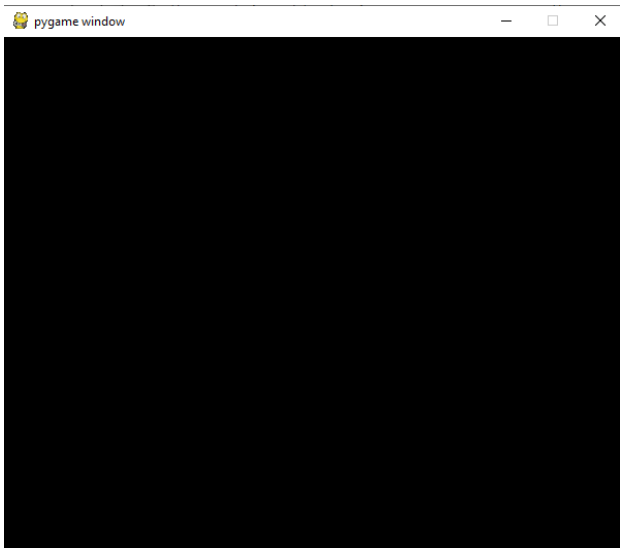
```
import pygame

pygame.init()
screen = pygame.display.set_mode((400,500))
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
done = False

while not done:
   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         done = True
   pygame.display.flip()
```



Here is the simple program of pygame which gives a basic idea of the syntax.

Let's understand the basic syntax of the above program line by line:

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

pygame.init() - This is used to initialize all the required module of the pygame.

pygame.display.set_mode((width, height)) - 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.

pygame.event.get() - 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.

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

pygame.display.flip() - 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.

Pygame Surface

Useful methods in each Surface object:

Surface((width, height))	constructs new Surface of given size
fill((red, green, blue))	paints surface in given color (rgb 0-255)
get_width(), get_height()	returns the dimensions of the surface
get_rect()	returns a Rect object representing the x/y/w/h bounding this surface

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

blit(surface, coords)	draws another surface onto this surface at the given coordinates

```
size = width, height = (32, 32)
empty_surface = pygame.Surface(size)
```

```
after changing any surfaces, must call display.update()
```

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.

To create a Surface you need at minimum it's size, which is a 2-element integer sequence of width and height, representing the size in pixels.

You can also pass additional arguments when creating a Surface to control bit depth, masks and additional features as per-pixel alpha and/or create the image in video memory.

You can use the **pygame.draw** module to draw shapes on the Surface, or fill it with a color by calling the Surface method **fill(color)**.

Pygame Adding Image

```
my_image = pygame.image.load(path_to_image)
```

```
my_image = pygame.image.load(path_to_image).convert()
```

```
my_image = pygame.image.load(path_to_image).convert_alpha()
```

More often than not you'd like to use your own images in a game (called sprites). Creating a Surface with your image on is as easy.

The path to the image can be relative or absolute. To improve performance it's usually wise to convert your image to the same pixel format as the screen. This can be done by calling the Surface method **convert()**.

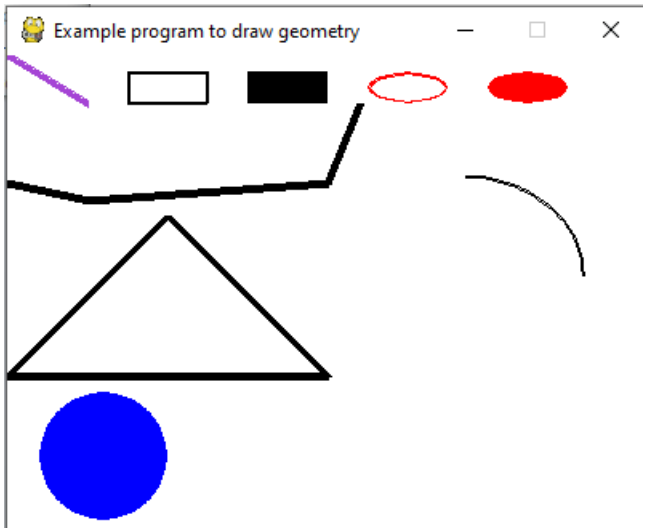
If your image contains transparency (alpha values) you just call the method **convert_alpha()**.

Drawing

screen.fill(white)	drawing the white background
pygame.draw.polygon(screen, green, ((146, 0), (291, 106), (236, 277), (56, 277), (0, 106)))	drawing the polygon
pygame.draw.line(screen, blue, (60, 60), (120, 60), 4)	drawing the lines

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

pygame.draw.circle(screen, blue, (300, 50), 20, 0)	drawing the circle
pygame.draw.ellipse(screen, red, (100, 150, 40,80), 1)	drawing the ellipse
pygame.draw.rect(screen,red, (200, 150, 100, 50))	drawing the rectangle



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.

Pygame Rect

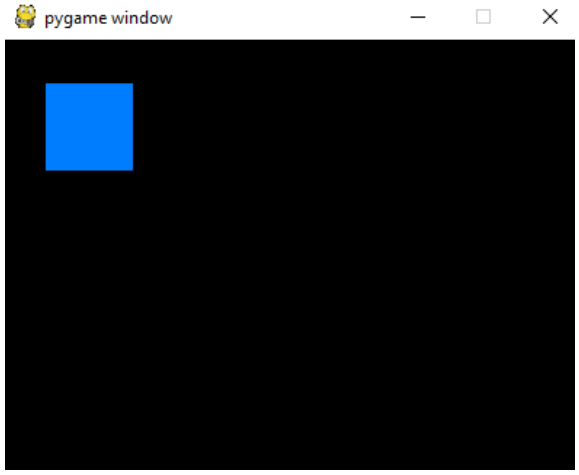
```
import pygame

pygame.init()
screen = pygame.display.set_mode((400, 300))
done = False

while not done:
   for event in pygame.event.get():
      if event.type == pygame.QUIT:
         done = True
   pygame.draw.rect(screen, (0, 125, 255), pygame.Rect(30, 30, 60, 60))

   pygame.display.flip()
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



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.

Let's create a Rectangle on the pygame window using the Rect.

After execution of the above code, it will display the rectangle on the pygame window.

Text and Font

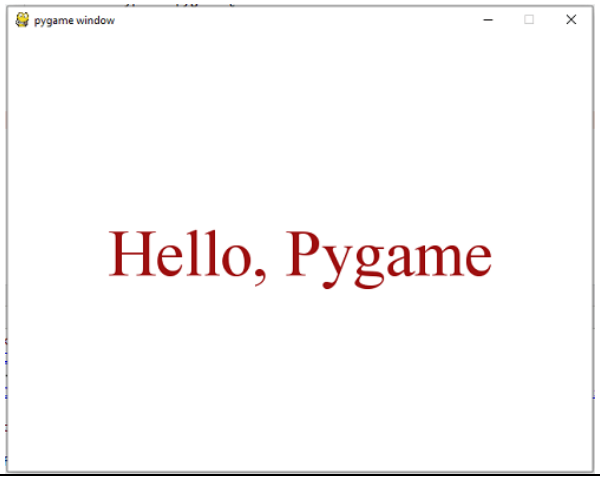
```
import pygame
pygame.init()
screen = pygame.display.set_mode((640, 480))
done = False

#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))

    screen.blit(text,(320 - text.get_width() // 2, 240 - text.get_height() // 2))

    pygame.display.flip()
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.




```
render(text, antialias, color, background=None)
size(bool)
set_bold(bool)
```

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.

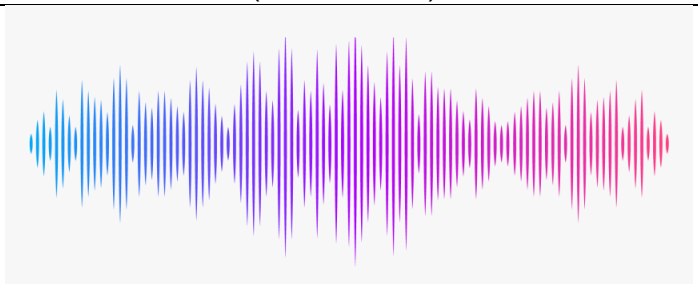
render()-This function is used to draw text on a new Surface. Pygame has no facility to draw text on the existing Surface. This creates a new Surface with the specified text render on it.

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.

set_bold()-This function is used for bold rending of text.

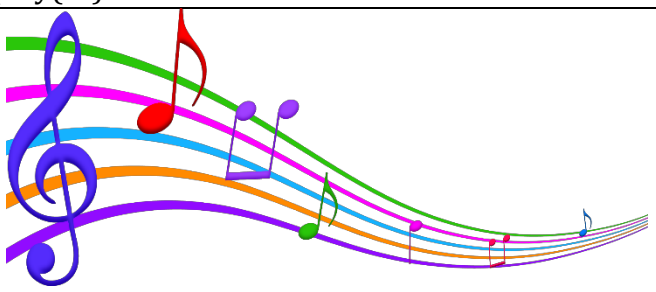
Music and Sound effects

```
crash_sound = pygame.mixer.Sound("crash.wav")
```



Sound

```
pygame.mixer.music.load('jazz.wav')
pygame.mixer.music.play(-1)
```



This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Video games are meant to immerse the player into a sort of virtual reality. We do this mainly visually, but it can make a massive difference and be a massive improvement to your game if you add sounds as well.

Sounds generally come in two major forms: Either "ambient" noise or as results of player actions. With PyGame, you get two choices: Music or Sounds.

The sound:

The above will assign the **crash.wav** sound file to play when we call **crash_sound** within PyGame's sound playing functionality. Notice the file is **.wav**.

PyGame can handle **.mp3** as well, but it is somewhat glitchy, and will work sometimes and other times it wont. For the safest bet, go with **.wav**.

The music:

The above code will play the music file indefinitely (though you can call it to stop). The **-1** signals PyGame to just play forever, but, if you put, say, a **5** in there, then the music would play once and **5** more times.

Handling Events

Constant name	Attributes
QUIT	none
ACTIVEEVENT	gain, state
KEYDOWN	unicode, key, mod
KEYUP	key, mod
MOUSEMOTION	pos, rel, buttons
MOUSEBUTTONUP	pos, button
MOUSEBUTTONDOWN	pos, button
JOYAXISMOTION	joy, axis, value
JOYBALLMOTION	joy, ball, rel
JOYHATMOTION	joy, hat, value
JOYBUTTONUP	joy, button
JOYBUTTONDOWN	joy, button
VIDEORESIZE	size, w, h
VIDEOEXPOSE	None
USEREVENT	code

Example

```
for event in pygame.event.get():
if event.type == pygame.QUIT:
# Close the program any way you want, or troll users who want to close your program.
raise SystemExit
```

The most essential part of any interactive application is the event loop. Reacting to events allows the user to interact with the application. Events are the things that can happen in a program, such as a

- mouse click,

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- mouse movement,
- keyboard press,
- joystick action

Pygame will register all events from the user into an event queue which can be received with the code **pygame.event.get()**. Every element in this queue is an **Event** object and they'll all have the attribute **type**, which is an integer representing what kind of event it is. In the pygame module there are predefined integer constants representing the type. Except for this attribute, events have different attributes.

Example.

To handle our events we simply loop through the queue, check what type it is (with the help of the predefined constants in the pygame module) and then perform some action. This code will check if the user has pressed the close button on the top corner of the display, and if so terminate the program.

Keyboard events

This code will check if the user has pressed w, a, s or d.

```
for event in pygame.event.get():
if event.type == pygame.QUIT: # Usually wise to be able to close your program.
    raise SystemExit
elif event.type == pygame.KEYDOWN:
if event.key == pygame.K_w:
    print("Player moved up!")
elif event.key == pygame.K_a:
    print("Player moved left!")
elif event.key == pygame.K_s:
    print("Player moved down!")
elif event.key == pygame.K_d:
    print("Player moved right!")
```

There are two types of key events in pygame: ***KEYDOWN*** and ***KEYUP***. These events have an attribute **key** which is an integer representing a key on the keyboard. The pygame module has predefined integer constants representing all the common keys. The constants are named with a capital **K**, an underscore and the name of the key. For example, <- is named **K_BACKSPACE**, a is named **K**_a and **F4** is namned **K_F4**.

Example.

Mouse events

Here's a short example using some of the attributes of each mouse event:

```
for event in pygame.event.get():
if event.type == pygame.QUIT: # Close your program if the user wants to quit.
    raise SystemExit
elif event.type == pygame.MOUSEMOTION:
    if event.rel[0] > 0: # 'rel' is a tuple (x, y). 'rel[0]' is the x-value.
        print("You're moving the mouse to the right")
    elif event.rel[1] > 0: # pygame start y=0 at the top of the display, so higher yvalues are
further down.
        print("You're moving the mouse down")
elif event.type == pygame.MOUSEBUTTONDOWN:
if event.button == 1:
```

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
        print("You pressed the left mouse button")
    elif event.button == 3:
        print("You pressed the right mouse button")
    elif event.type == pygame.MOUSEBUTTONUP:
        print("You released the mouse button")
```

There are three types of mouse events in pygame **MOUSEMOTION**, **MOUSEBUTTONDOWN** and **MOUSEBUTTONUP**.

Pygame will register these events when a display mode has been set.

MOUSEMOTION is received when the user moves his or her mouse in the display. It has the attributes **buttons, pos** and **rel**.

• **buttons** is a tuple representing if the mouse buttons **(left, mouse-wheel, right)** are being pressed or not.

• **pos** is the absolute position **(x, y)** of the cursor in pixels.

• **rel** is the position relative to the previous position **(rel_x, rel_y)** in pixels.

MOUSEBUTTONDOWN and **MOUSEBUTTONUP** is received when the user presses or releases a mouse button.

They have the attributes **button** and **pos**.

• **button** is an integer representing the button being pressed. **1** for left button, **2** for mousewheel and **3** for right button.

• **pos** is the absolute position of the mouse **(x, y)** when the user pressed the mouse button.

REFERENCES

1. Link: [https://www.javatpoint.com/pygame]
2. Link: [https://www.javatpoint.com/pygame]
3. Unit 9. Special thanks to Roy McElmurry, John Kurkowski, Scott Shawcroft, Ryan Tucker, Paul Beck for their work. Except where otherwise noted, this work is licensed under: http://creativecommons.org/licenses/by-nc-sa/3.0
4. Free unaffiliated eBook created from Stack Overflow contributors
5. Link: [https://pythonprogramming.net/adding-sounds-music-pygame/]

This document has been produced with the support of the EUROPEAN COMMISSION under the ERASMUS+ Programme, KA2 – Capacity Building in the Field of Higher Education: 598092-EPP-1-2018-1-BG-EPPKA2-CBHE-SP.
It reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.