| EX.NO:11 | EXPLORING PYGAME TOOL |
|---|---|
| DATE: | |

**AIM:**

To explore pygame tool

**Python Pygame (Game Development Library)**

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. Python has vast libraries for various fields such as **Machine Learning(Numpy, Pandas, Matplotlib), Artificial intelligence (Pytorch, TensorFlow), and Game development (Pygame,Pyglet)**.

**Pygame**

- 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.

**Prerequisites for Pygame**:

Before learning about pygame, we need to understand what kind of game we want to develop.

- To learn pygame, it is required to have basic knowledge of Python.

**Pygame Installation**

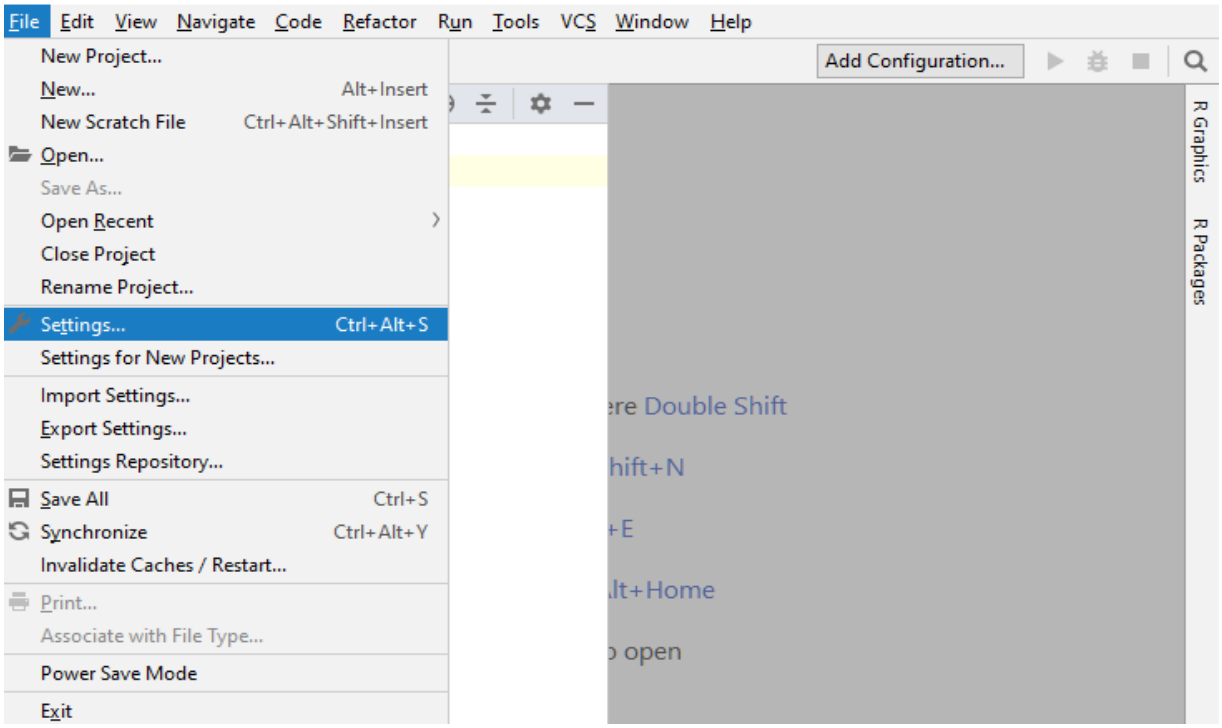
Install pygame in Windows

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.

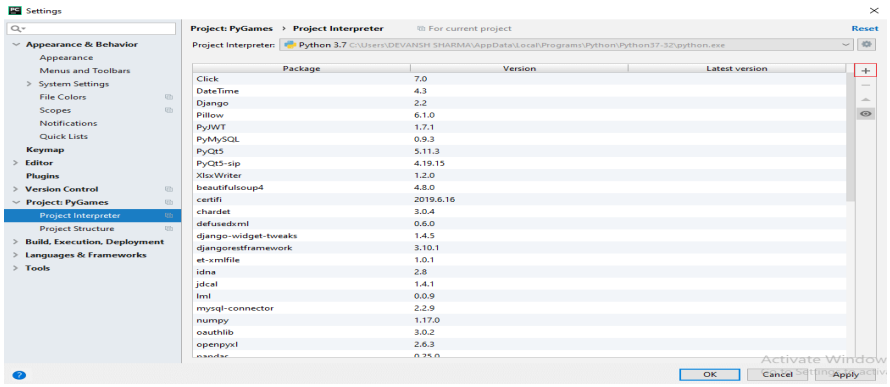
**1.Installing through pip:** The good way to install Pygame is with the pip tool (which is what python uses to install packages). The command is the following:
**py -m pip install -U pygame --user**

**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:

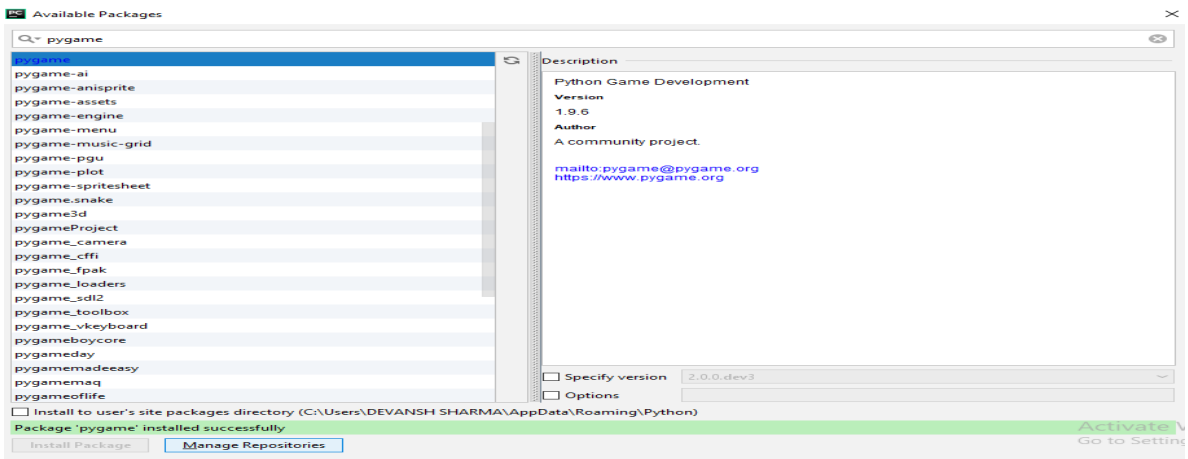
- Open the **File** tab and click on the **Settings** option.




- 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.



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

1. **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.


**Introduction to Python Pygame :**

Programming via gaming is a teaching tool nowadays. Gaming includes logics, physics, math and AI sometimes. Python language uses pygame to carry out the programming. Pygame is the best module so far, helping in game development. Pygame is a module used to build games that are usually 2D. Knowing Pygame proficiently can take you far in developing extreme ranges of games or big games.

Syntax and Parameters of Python Pygame :

Below are the syntax and parameters:

**Syntax**

```
import pygame
from pygame.locals import *
```

Pygame

- 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.

**Parameters**

- importing pygame module followed by accessing key coordinates easily from locals
- From there, you write your game code where you want to insert objects movable, sizeable, etc.

Then initialize the pygame module.

**Examples to Implement Python Pygame**

Below are the examples:

**1. initializing the pygame**

pygame.init()

This is a function call. Before calling other pygame functions, we always call this first just after importing the pygame module. If you see an error that says font not initialized, go back and check if you forgot pygame.init() insertion at the very start.

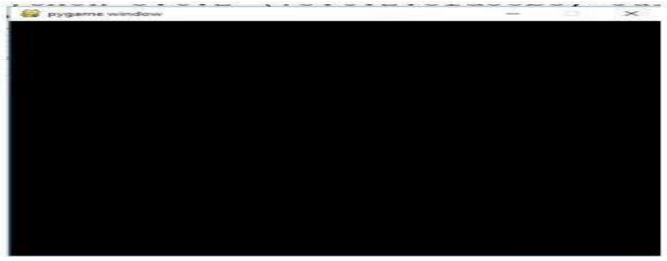
**2. Surface Object**

pygame.display.set_mode((500, 500))

This function call returns the pygame.Surface object. The tuple within tells the pixel parameters about width and height. This creates an error if not done properly, which says argument 1 must be 2 items sequences, not int.

**Code:**

```
import pygame
from pygame.locals import *
pygameinit()
display_window = pygame.display.set_mode((400, 300))
pygame.display.set_caption('Hello World!')
while True:
for event in pygame.event.get():
if event.type == QUIT:
pygame.quit()
sys.exit()
pygame.display.update()
```



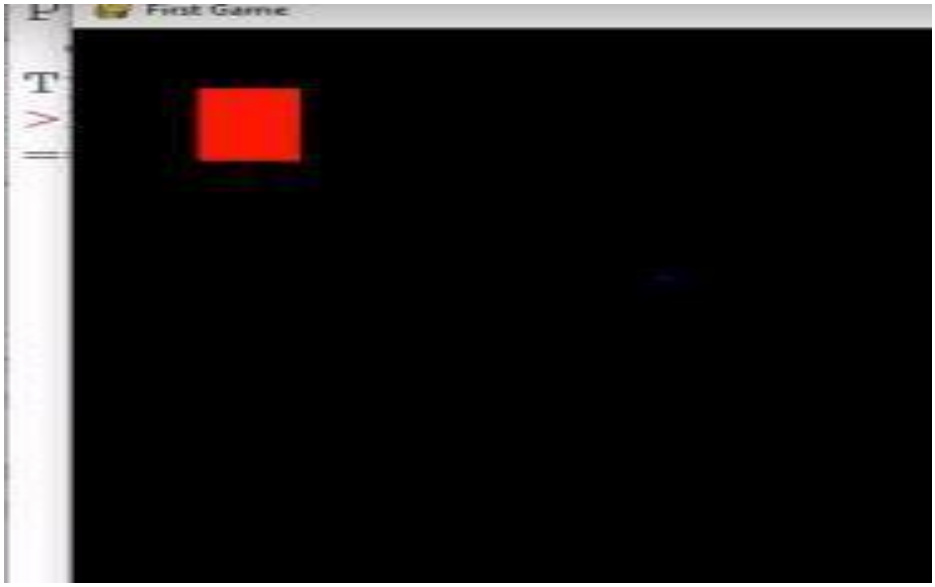
#Let’s make a character move on the screen.

**Code:**

```
import pygame
pygame.init()
win = pygame.display.set_mode((500, 500))
pygame.display.set_mode = ("First game")
x = 50
y = 50
width = 40
height = 60
vel = 5
run = True
while run:
pygame.time.delay(100)
for event in pygame.event.get():
if event.type == pygame.QUIT:
run = False
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
x -= vel
if keys[pygame.K_RIGHT]:
x += vel
if keys[pygame.K_UP]:
y -= vel
if keys[pygame.K_DOWN]:
y += vel
win.fill ((0,0,0))
pygame.draw.rect(win, (255, 0, 0),(x, y, widty, height))
pygame.display.update()
pygame.quit()
```

- First, we initialize the pygame module by importing it.
- Now, we give the parameters for the window size of the game we are constructing.
- Naming the game to the first game, which is the name of the window.
- Now, creating a character, it needs parameters. Let’s say we create a rectangle. A rectangle needs height, width, x and y coordinates to be placed in the window, the velocity with which it should move across the window.

**Output:** This is how the output seems for creating a character, that is, in this case, a rectangle.




- Then we need to start writing out the main loop, which considers the character movement. In this program, the main loop is going to check for its collision, its mouse events; it's also going to check if you hit something. This is one of the simple ways to do it.
- Now we make a run variable. In the loop, we are going to check the collision. Giving the loop a time delay is going to help you by delaying the things that happen real quick in the window. This regards kind od a clock in pygame. You cannot normally import the clock in pygame, but this is the easy way to do it.
- For checking an event: Events in pygame are anything that the user causes. Like, moving the mouse in general, accessing the computer to create files. So, to check for the events, we happened to create a loop check for events.
- Then we draw a rectangle that is movable, which is a surface on the pygame window. All the colors in pygame are in RGB, so 255 is red and the giving in the defined parameters width, height, x, y coordinates. Then we update the window, which displays us a rectangle at those particular coordinates and parameters.

**PyGame Concepts**

As `pygame` and the SDL library are portable across different platforms and devices, they both need to define and work with abstractions for various hardware realities. Understanding those concepts and abstractions will help you design and develop your own games.

### Initialization and Modules

The `pygame` library is composed of a number of Python constructs, which include several different **modules**. These modules provide abstract access to specific hardware on your system, as well as uniform methods to work with that hardware. For example, `display` allows uniform access to your video display, while `joystick` allows abstract control of your joystick.

After importing the `pygame` library in the example above, the first thing you did was initialize PyGame using `pygame.init()`. This function calls the separate `init()` functions of all the included `pygame` modules. Since these modules are abstractions for specific hardware, this initialization step is required so that you can work with the same code on Linux, Windows, and Mac.

### Displays and Surfaces

In addition to the modules, `pygame` also includes several Python **classes**, which encapsulate non-hardware dependent concepts. One of these is the `Surface` which, at its most basic, defines a rectangular area on which you can draw. `Surface` objects are used in many contexts in `pygame`. Later you'll see how to load an image into a `Surface` and display it on the screen.

In `pygame`, everything is viewed on a single user-created `display`, which can be a window or a full screen. The display is created using `.set_mode()`, which returns a `Surface` representing the visible part of the window. It is this `Surface` that you pass into drawing functions like `pygame.draw.circle()`, and the contents of that `Surface` are pushed to the display when you call `pygame.display.flip()`.

### Images and Rects

Your basic `pygame` program drew a shape directly onto the display's `Surface`, but you can also work with images on the disk. The `image` module allows you to load and save images in a variety of popular formats. Images are loaded into `Surface` objects, which can then be manipulated and displayed in numerous ways.

As mentioned above, `Surface` objects are represented by rectangles, as are many other objects in `pygame`, such as images and windows. Rectangles are so heavily used that there is a special `Rect` class just to handle them. You'll be using `Rect` objects and images in your game to draw players and enemies, and to manage collisions between them.

## Basic Game Design

Before you start writing any code, it's always a good idea to have some design in place. Since this is a tutorial game, let's design some basic gameplay for it as well:

- The goal of the game is to avoid incoming obstacles:
  - The player starts on the left side of the screen.
  - The obstacles enter randomly from the right and move left in a straight line.
- The player can move left, right, up, or down to avoid the obstacles.
- The player cannot move off the screen.
- The game ends either when the player is hit by an obstacle or when the user closes the window.

When he was describing software projects, a former colleague of mine used to say, "You don't know what you do until you know what you don't do." With that in mind, here are some things that won't be covered in this tutorial:

- No multiple lives
- No scorekeeping
- No player attack capabilities
- No advancing levels
- No boss characters

You're free to try your hand at adding these and other features to your own program.

Let’s get started!

Importing and Initializing PyGame

After you import pygame, you’ll also need to initialize it. This allows pygame to connect its abstractions to your specific hardware:

```
# Import the pygame module
 import pygame
 # Import pygame.locals for easier access to key coordinates
 # Updated to conform to flake8 and black standards
 from pygame.locals import (K_UP,K_DOWN,K_LEFT, K_RIGHT,K_ESCAPE,KEYDOWN,
 QUIT)
# Initialize pygame
pygame.init()
```

The pygame library defines many things besides modules and classes. It also defines some local constants for things like keystrokes, mouse movements, and display attributes. You reference these constants using the syntax pygame.<CONSTANT>. By importing specific constants from pygame.locals, you can use the syntax <CONSTANT> instead. This will save you some keystrokes and improve overall readability.

## Setting Up the Display

Now you need something to draw on! Create a screen to be the overall canvas:

```
# Import the pygame module
importpygame
# Import pygame.locals for easier access to key coordinates
# Updated to conform to flake8 and black standards
from pygame.locals import (K_UP,K_DOWN,K_LEFT, K_RIGHT,K_ESCAPE,KEYDOWN,
 QUIT)
# Initialize pygame
pygame.init()
# Define constants for the screen width and height
SCREEN_WIDTH = 800
SCREEN_HEIGHT = 600
# Create the screen object
# The size is determined by the constant SCREEN_WIDTH and SCREEN_HEIGHT
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
```

You create the screen to use by calling pygame.display.set_mode() and passing a tuple or list with the desired width and height. In this case, the window is 800x600, as defined by theconstants SCREEN_WIDTH and SCREEN_HEIGHT on lines 20 and 21. This returns a Surface which represents the inside dimensions of the window. This is the portion of the window you can control, while the OS controls the window borders and title bar.

If you run this program now, then you’ll see a window pop up briefly and then immediately disappear as the program exits.

Setting Up the Game Loop

Every game from Pong to Fortnite uses a game loop to control gameplay. The game loop does four very important things:

1. Processes user input
2. Updates the state of all game objects
3. Updates the display and audio output
4. Maintains the speed of the game

Every cycle of the game loop is called a **frame**, and the quicker you can do things each cycle, the faster your game will run. Frames continue to occur until some condition to exit the game is met. In your design, there are two conditions that can end the game loop:

1. The player collides with an obstacle. (You'll cover collision detection later.)
2. The player closes the window.

The first thing the game loop does is process user input to allow the player to move around the screen. Therefore, you need some way to capture and process a variety of input. You do this using the pygame event system.