PROGRAMACIÓN AVANZADA CON PYTHON

(CEFIRE CTEM)

Introducción a PyGame

Autora: María Paz Segura Valero (segura_marval@gva.es)

CONTENIDO

1. Introducción

A lo largo de las sesiones anteriores hemos ido adquiriendo los conocimientos básicos para crear en Python programas de propósito general. En esta sesión daremos un paso más allá y nos adentraremos en el mundo de la creación de juegos.

Para ello, utilizaremos la librería **PyGame**. Esta librería ofrece módulos con funciones que permiten manejar *gráficos*, *sonidos*, *eventos del ratón* y más cosas con Python.

En este tipo de aplicaciones necesitamos que el programa esté pendiente de todas y cada una de las acciones que realiza el usuario (por ejemplo: clic del botón izquierdo del ratón, movimiento del cursor del ratón hacia la derecha, doble clic sobre un objeto del escenario, etc) y que reaccione a cada una de ellas sin haber realizado una pregunta previa.

Este tipo de programas pertenecen a la **programación dirigida por eventos** y responden a los eventos que se van produciendo en el programa sin que exista un orden preestablecido entre ellos.

En este documento conoceremos los fundamentos básicos de la creación de juegos con PyGame y crearemos un esqueleto de programa que nos sirva de base para crear distintos juegos en Python.

2. Instalación y uso de la librería PyGame

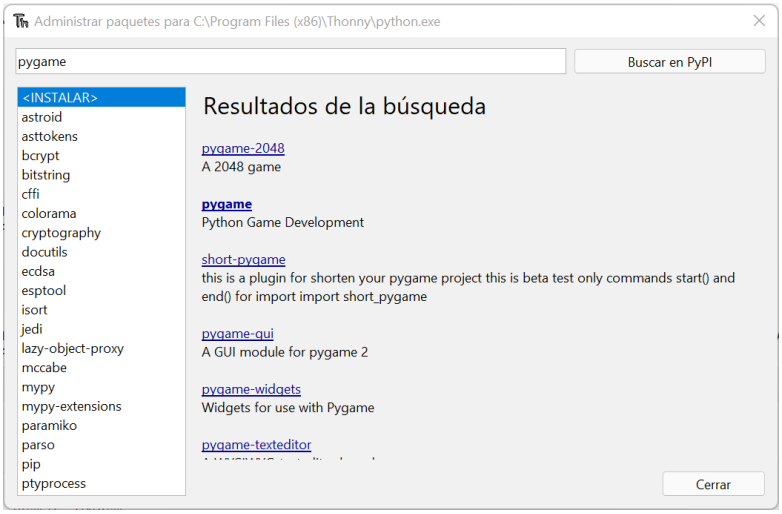
Para instalar la librería PyGame utilizaremos una herramienta de Python llamada **pip** que nos permite instalar paquetes de una manera sencilla.

Los pasos que hay que seguir se indican a continuación.

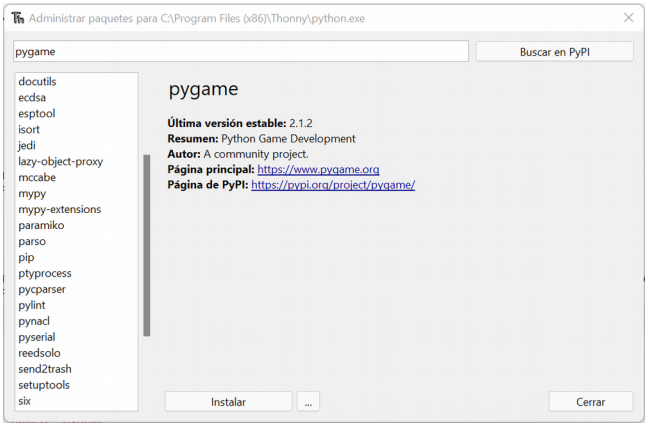
2.1. Instalación usando el editor Thonny

Si vas a ejecutar los programas a través del editor Thonny, solo será necesario que añadas la extensión de **PyGame** siguiendo estos pasos:

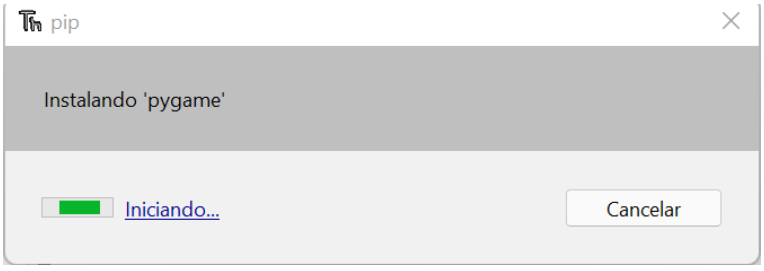
1. Seleccionamos la opción de menú **Herramientas | Gestionar paquetes** y buscamos el paquete **pygame**:



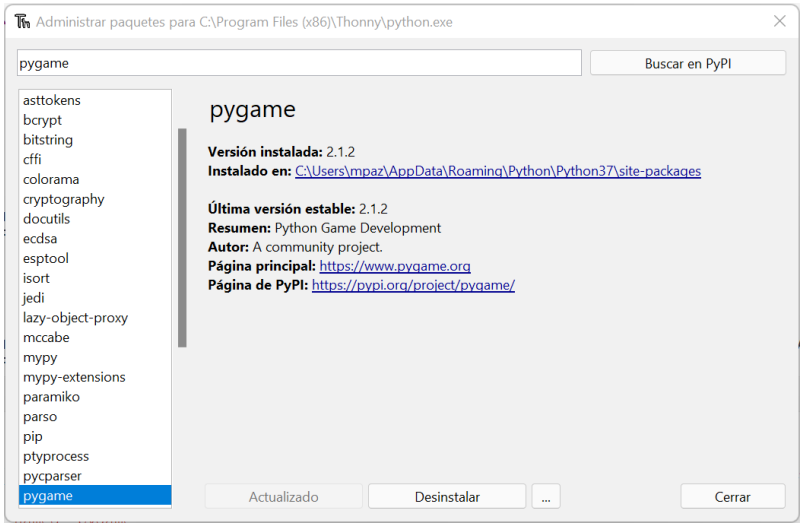
2. Pulsamos sobre el enlace **pygame** y aparecerá la información del paquete:



3. Pulsaremos el botón **Instalar** y comenzará la instalación del paquete.



4. Cuando haya acabado la instalación, aparecerá información de la misma.



Hay que tener en cuenta que esta instalación la estamos haciendo para trabajar dentro del editor **Thonny**. Si queremos ejecutar los programas desde fuera, deberemos realizar la instalación tal y como se detalla en el siguiente apartado.

2.2. Instalación si no utilizas el editor Thonny

Si quieres ejecutar videojuegos hechos con Python y PyGame directamente desde el sistema operativo, deberás seguir los pasos que se describen en este apartado.

2.2.1. Instalación en Windows

Abrimos una *terminal de comandos* y ejecutamos las siguientes instrucciones:

1. Actualizamos la herramienta **pip**:

   ```
   python.exe -m pip install --upgrade pip
   ```

2. Podemos comprobar la versión que hemos instalado:

   ```
   pip --version
   ```

3. Ahora, instalamos la librería **pygame**:

   ```
   pip install pygame
   ```

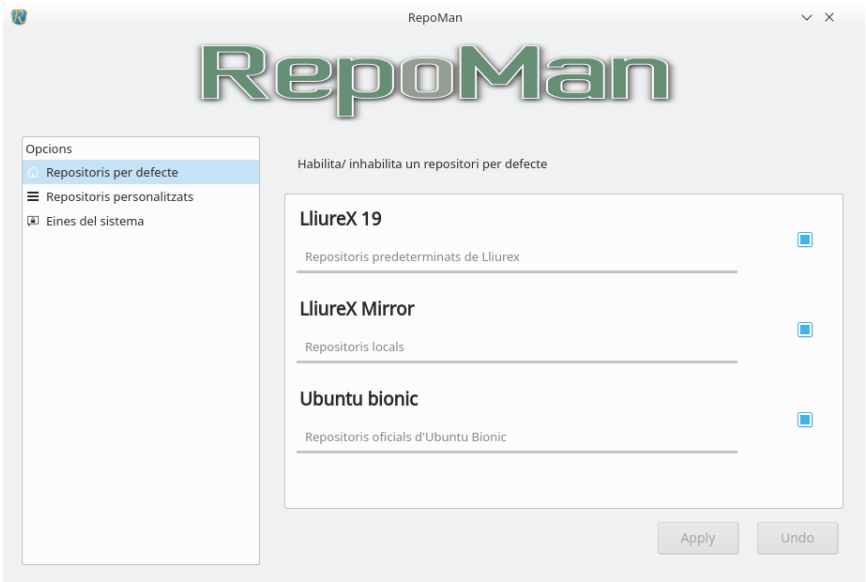
4. Para comprobar si se ha instalado correctamente podemos ejecutando uno de sus programas de ejemplo o probar a importar la librería desde python3

   ```
   python -m pygame.examples.aliens
   ```

2.3. Instalación en Lliurex 19

Abrimos una *terminal de comandos* y ejecutamos las siguientes instrucciones:

1. Actualizamos los repositorios de Lliurex. Para ello, abre la herramienta **RepoMan**, accede a los *Repositorios por defecto* y asegúrate de que están seleccionados los tres:



2. Ahora vamos a instalar la herramienta **pip**:

```
sudo apt update
sudo apt install python3-pip
```

3. Podemos comprobar la versión que hemos instalado:

```
pip3 --version
```

4. Un vez tengamos instalada la herramienta **pip3**, la actualizaremos con el siguiente comando:

```
pip3 install -U pip
```

   De esta forma, obtendremos la versión más reciente de la herramienta y, a partir de ahora, podremos utilizar directamente el nombre `pip` en lugar de `pip3`.

5. Ahora instalamos la librería **PyGame**:

```
python3 -m pip install -U pygame –user
```

6. Para comprobar si se ha instalado correctamente podemos ejecutar uno de sus programas de ejemplo.

```
python3 -m pygame.examples.aliens
```

2.4. Uso de la librería PyGame

Para poder utilizar los elementos de la librería PyGame en un programa Python, deberemos importarla. Por ejemplo, así:

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

El módulo `pygame.locals` contiene muchas constantes y variables que vamos a utilizar frecuentemente en el programa. Importándolo de esta manera no será necesario anteponer el texto `pygame.` delante de cada elemento. Sí será necesario para el resto de elementos de la librería.

3. Estructura básica de un juego

Los programas con los que se desarrollan juegos en Python suelen utilizar una estructura muy similar a la que se muestra a continuación:

```
inicializar()
mientras ejecutando():
        manejar_eventos()
        logica_juego()
        actualizar_pantalla()
acabar()
```

Nada más comienza el juego, en `inicializar()`, se carga la imagen del escenario, los audios que se deben utilizar, los niveles del juego y cualquier dato que se necesite para su funcionamiento. Éste sería el momento en el que se accedería a la base de datos para recuperar información del estado de cada uno de los jugadores, por ejemplo.

A continuación, entramos en un bucle del que saldremos sólo cuando se reciba un evento de salida: o bien porque el jugador decida acabar el juego o bien porque ha sucedido alguna excepción que nos obliga a finalizar el programa. La función `ejecutando()` devolverá un valor booleano que indicará si debemos continuar o no.

Dentro del bucle realizaremos varias tareas:

`manejar_eventos()`	Gestionar los eventos generados por el usuario o el sistema. Por ejemplo: clic del botón izquierdo del ratón, movimiento del cursor del ratón, tecla ESC pulsada.

`logica_juego()`	Comprobar las consecuencias que tienen las acciones del jugador sobre el escenario y los elementos del juego. Por ejemplo: disminuir el nivel de vida de un personaje, controlar si se ha producido un choque contra una obstáculo, aumentar los puntos del jugador.
`actualizar_pantalla()`	Realizar cambios en el escenario del juego como resultado de las acciones del jugador. Por ejemplo: cambiar el fondo del escenario si ha subido de nivel o mover el personaje hacia su derecha.

Cuando se sale del bucle, en la función `acabar()`, se realizan las acciones necesarias para dar por finalizado el juego. Por ejemplo: liberar recursos del sistema o actualizar la base de datos con los nuevos puntos del jugador.

Evidentemente, dependiendo del tipo de juego, la estructura del programa puede variar. Sin embargo, puede ser un buen punto de partida para crear juegos sencillos, como los que vamos a tratar en esta unidad didáctica.

Vamos a ver cómo sería la traducción de esta estructura básica[1] en el lenguaje Python utilizando la librería PyGame. Iremos añadiendo elementos y explicándolos poco a poco.

4. Características del juego

El programa que vamos a presentar sólo muestra una imagen de fondo y detecta cuando se presiona el botón de cierre de la ventana del juego (evento QUIT). Aunque es un programa muy sencillo, nos sirve para presentar la estructura básica en la que basarnos para programar juegos más complejos.

Si quieres probar su funcionamiento, descarga del aula virtual los siguientes ficheros y ubícalos en la misma carpeta de tu ordenador:

`game.py`	Código del programa en Python
`pygame.png`	Imagen de fondo que mostraremos en la pantalla

1 Fuente: https://pythonspot.com/game-development-with-pygame/

4.1. Primera aproximación: inicio de la ejecución

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

2  class App:
       # Definición de atributos (variables) de la aplicación
       ...
       # Definición de métodos (funciones) de la aplicación
       ...
       def on_execute(self):
           ...

3  # Programa principal
   if __name__ == "__main__" :
       theApp = App()
       theApp.on_execute()
```

Lo primero que hace el programa (1) es importar los elementos necesarios de la librería PyGame.

A continuación, se define la clase `App` (2) que gestionará todas la variables y eventos del juego. Esta clase también proporcionará las funciones necesarias para manejar el contenido de la pantalla.

Por último, el programa principal (3) crea una instancia (objeto) de la clase `App`, lo almacena en la variable `theApp` y arranca su ejecución con la función `on_execute()`.

A continuación añadirems contenido a cada una de estas partes.

4.2. Segunda aproximación: inicialización de los atributos

```
   import pygame
   from pygame.locals import *

   class App:
1      # Definición de atributos (variables) de la aplicación
       windowWidth = 640
       windowHeight = 480
       x = 10
       y = 10

2      def __init__(self):
           self._running = True
           self._display_surf = None
           self._image_surf = None
```

```
    # Definición de métodos (funciones) de la aplicación
    def on_execute(self):
            ...

# Programa principal
if __name__ == "__main__" :
    theApp = App()
    theApp.on_execute()
```

3

Cuando se crea el objeto `theApp` de la clase `App` se da valor a sus atributos de dos formas:

- En el apartado (1) se da valor a cuatro atributos de clase, que tendrán el mismo valor para todos los objetos de la clase `App`.
- En el apartado (2) se utiliza el método `__init__()` para dar valores iniciales a los atributos del objeto `theApp`. Este método se ejecuta en el momento (3), cuando se está creando una instancia de la clase `App` y justo antes de asignar el nuevo objeto a la variable `theApp`.

En otras palabras, en el momento de crear el objeto `theApp` y antes de que lancemos su método `on_execute()`, se dan valores iniciales a los atributos del objeto `theApp`.

La palabra `self` hace referencia al objeto en cuestión con el que estamos trabajando. En este caso, se trataría del objeto `theApp` que estamos creando.

4.3. Tercera aproximación: empezar y acabar

```
import pygame
from pygame.locals import *

class App:
    # Definición de atributos (variables) de la aplicación
    windowWidth = 640
    windowHeight = 480
    x = 10
    y = 10

    def __init__(self):
        self._running = True
        self._display_surf = None
        self._image_surf = None

    # Definición de métodos (funciones) de la aplicación
    def on_init(self):
        pygame.init()
        self._running = True
```

```
        self._display_surf =
            pygame.display.set_mode((self.windowWidth,
                  self.windowHeight), pygame.HWSURFACE)
        self._image_surf =pygame.image.load("pygame.png").convert()

    def on_cleanup(self):
        pygame.quit()

    def on_execute(self):
        self.on_init()
        ...
        self.on_cleanup()

# Programa principal
if __name__ == "__main__" :
    theApp = App()
    theApp.on_execute()
```

Dentro del método `on_execute()`, encontramos llamadas a dos métodos de la clase:

`on_init()`	Este método se encarga de varias cosas: • inicializar todos los módulos contenidos en la librería PyGame (`pygame.init()`) • indicar que estamos ejecutando el juego (`self._running`) • inicializar la pantalla indicando el tamaño de la misma y el tipo de vista. (`self._display_surf`) • indicar la imagen de fondo de pantalla (`self._image_surf`)[2]
`on_cleanup()`	Este método cierra todos los recursos asociados a la aplicación. En este caso sólo llama a `pygame.quit()`, que se encarga de desactivar el módulo PyGame, pero podría contener más código.

4.4. Cuarta aproximación: manejo de eventos

```
import pygame
from pygame.locals import *

class App:
    # Definición de atributos (variables) de la aplicación
    windowWidth = 640
    windowHeight = 480
```

2 La imagen debe encontrarse en la misma carpeta que el programa Python.

```
    x = 10
    y = 10

    def __init__(self):
        self._running = True
        self._display_surf = None
        self._image_surf = None

    # Definición de métodos (funciones) de la aplicación
    def on_init(self):
        pygame.init()
        self._running = True
        self._display_surf =
            pygame.display.set_mode((self.windowWidth,
                  self.windowHeight), pygame.HWSURFACE)
        self._image_surf =pygame.image.load("pygame.png").convert()

    def on_event(self, event):
        if event.type == QUIT:
            self._running = False

    def on_cleanup(self):
        pygame.quit()

    def on_execute(self):
        self.on_init()
        while( self._running ):
            for event in pygame.event.get():
                self.on_event(event)
            ...
        self.on_cleanup()

# Programa principal
if __name__ == "__main__" :
    theApp = App()
    theApp.on_execute()
```

1

2

Mientras no acabe el juego (`self_running == True`), en el punto (1), el programa va atendiendo los eventos que se van produciendo. Estos eventos pueden ser las acciones que realiza el jugador Por ejemplo: mover el ratón, pulsar sobre un personaje del escenario, pulsar la tecla ESC, etc.

Para ello, el programa recorre la lista de eventos (2) que devuelve la función `pygame.event.get()` y, para cada uno de ellos, llama al método `self.on_event()` para gestionarlo. En este caso, sólo se da respuesta al evento `QUIT` (1) que cambia el valor del atributo `self._running` a `False` y provoca que salgamos del bucle del juego.

4.5. Quinta aproximación: lógica del juego

```
import pygame
from pygame.locals import *

class App:
    # Definición de atributos (variables) de la aplicación
    windowWidth = 640
    windowHeight = 480
    x = 10
    y = 10

    def __init__(self):
        self._running = True
        self._display_surf = None
        self._image_surf = None

    # Definición de métodos (funciones) de la aplicación
    def on_init(self):
        pygame.init()
        self._running = True
        self._display_surf =
            pygame.display.set_mode((self.windowWidth,
                  self.windowHeight), pygame.HWSURFACE)
        self._image_surf =pygame.image.load("pygame.png").convert()

    def on_event(self, event):
        if event.type == QUIT:
            self._running = False


    def on_loop(self):
        pass

    def on_cleanup(self):
        pygame.quit()

    def on_execute(self):
        self.on_init()
        while( self._running ):
            for event in pygame.event.get():
                self.on_event(event)
            self.on_loop()
            ...
        self.on_cleanup()

# Programa principal
if __name__ == "__main__" :
    theApp = App()
    theApp.on_execute()
```

Después de dar respuesta a los eventos detectados, y mientras sigamos en ejecución (`self_running == True`), el método `self.on_loop()` determina los cambios que se han producido en la lógica del juego y hace las modificaciones oportunas. Por ejemplo: si hemos chocado con un obstáculo, si hemos agotado la vida del personaje, si hemos conseguido un cuatro en línea, etc.

Ésta es la parte de la estructura del programa que más cambiará de un programa a otro, ya que depende de las reglas básicas del juego en cuestión. En este caso, no vamos a hacer nada (`pass`) pero dejamos preparada la estructura del programa para cuando vayamos a implementar un juego completo.

5. Sexta aproximación: actualizar la pantalla

```
import pygame
from pygame.locals import *

class App:
    # Definición de atributos (variables) de la aplicación
    windowWidth = 640
    windowHeight = 480
    x = 10
    y = 10

    def __init__(self):
        self._running = True
        self._display_surf = None
        self._image_surf = None

    # Definición de métodos (funciones) de la aplicación
    def on_init(self):
        pygame.init()
        self._running = True
        self._display_surf =
            pygame.display.set_mode((self.windowWidth,
                  self.windowHeight), pygame.HWSURFACE)
        self._image_surf =pygame.image.load("pygame.png").convert()

    def on_event(self, event):
        if event.type == QUIT:
            self._running = False

    def on_loop(self):
        pass

    def on_render(self):
        self._display_surf.blit(self._image_surf,(self.x,self.y))
        pygame.display.flip()
```

```
    def on_cleanup(self):
        pygame.quit()

    def on_execute(self):
        self.on_init()
        while( self._running ):
            for event in pygame.event.get():
                self.on_event(event)
            self.on_loop()
            self.on_render()
        self.on_cleanup()

# Programa principal
if __name__ == "__main__" :
    theApp = App()
    theApp.on_execute()
```

Por último, y después de hacer los cálculos pertinentes en el programa, mostramos por pantalla el nuevo aspecto del escenario y/o los personajes u objetos involucrados en el juego. Esto lo conseguimos ejecutando el método `self.on_render()` que contiene dos llamadas a funciones:

`self._display_surf.blit (self._image_surf, (self.x,self.y))`	Coloca una imagen en una posición concreta de la pantalla pero no la muestra hasta que no se ejecuta la instrucción `flip`.
`pygame.display.flip()`	Actualiza el aspecto de la pantalla con los cambios.

Puedes ver el ejemplo completo en el siguiente fichero: ***game.zip***

6. Fuentes de información

- Librería PyGame: https://github.com/pygame/pygame