pygame lab

This lab is guided. The code is available on D2L. The lab is not graded, but you should complete all the exercises.

Pygame concepts

Install pygame

- `pygame` can be installed with `pip`. Make sure your virtual environment is active!

Using pygame

Pygame has many features. The ones we are going to use are:

- management of the graphical interface (window, images, graphics)
- interaction with user input (mouse, keyboard)
- management of time (FPS and event loop)

In order to be used, the pygame library must be initialized:

```python
import pygame

pygame.init()
```

You can then open the main window. The following command will open a window with dimensions 500x500 (pixels):

```python
window = pygame.display.set_mode((500, 500))
```

Pygame `Surface`

A very important concept in pygame is the `Surface`. All graphical elements are drawn on a surface. The main window is a surface too. You can draw and paint surfaces, and display images on them.

The following will create a surface (200x200 pixels), and fill it with white:

```python
surface = pygame.Surface((200, 200))
surface.fill((255, 255, 255))
```

⚠️ The arguments are tuples!

Colors in pygame

The default color space in pygame is RGB (**R**ed **G**reen **B**lue). Colors can be represented as 3-tuples - each value must be between 0 and 255.

RGB value	Color
0, 0, 0	black
255, 255, 255	white
255, 0, 0	red
0, 255, 0	green
0, 0, 255	blue
255, 165, 0	orange
238, 130, 238	violet

Transparency

Pygame has several ways to manage transparency. The easiest is to define a "color key": all pixels of that color will not be blitted (= will be transparent). Example:

```
my_surface.set_colorkey((0, 0, 0)) # Will make black transparent
```

Drawing shapes

You can draw shapes on a `Surface`: use `pygame.draw`. For instance, you can draw a circle of radius 80 pixels with the center coordinates (20, 30):

```
surface = pygame.Surface((200, 200))
pygame.draw.circle(surface, (0, 0, 0), (20, 30), 50)
#                            color     center   radius
```

pygame coordinates

Coordinates in pygame refer to the **pixel** coordinates. There are no half-pixels in pygame: they are integers (x, y). The coordinate system starts from the top-left of the surface: (0, 0).

- When you go right, `x` goes up.

- When you go down, `y` goes up.

If a surface is 200 pixels wide by 100 pixels high: `surface = pygame.Surface((200, 100))`, then:

- (0, 0) is (always) the top-left corner
- (200, 0) is the top-right corner
- (200, 100) is the bottom-right corner
- (0, 100) is the bottom-left corner
- (100, 50) is the center

Using `Rect` objects

It can be easier to use `Rect` instances to address surface locations.

```
surface = pygame.Surface((200, 200))
rect = surface.get_rect()
print(rect.left, rect.topleft, rect.bottom, rect.right, rect.bottomright)
```

Layering surfaces with pygame

pygame allows you to display a surface on top of another surface: it's called **blitting**. We can display our circle shape in the main window:

```
window.blit(surface, (100, 200))
```

This will blit the surface (with a circle drawn on it) onto the main screen. pygame manage transparency. There are different ways of dealing with transparency - one of them is to set the `color key` to black (the default background).

```
surface.set_colorkey((0, 0, 0))
window.blit(surface, (100, 200))
```

Updating the display

Once all surfaces have been blitted, you can update pygame's display. This is done with: `pygame.display.update()`. This method is usually called at regular intervalls. The number of times it is called is essentially the FPS (frames per second).

Exercise 1: `L1_basic_shapes.py`

- Complete the code in `L1_basic_shapes.py`. Create a circle and a rectangle.
- Fill them with different colors.
- Try out several combinations!
- Feel free to use more shapes from `pygame.draw`
- https://www.pygame.org/docs/ref/draw.html

Understand the event loop

Pygame exposes an event-driven interface. It means that pygame runs as its own "controller", managing user input (including keyboard and mouse). Every time the user performs an action while pygame is running, it creates an `event`. The developers can regularly inspect the event list, and adjust the behaviour of the game.

> You **must** manage the event loop, or your Pygame program will take all resources on your computer and be very hard to use.

Polling pygame event loop

The typical structure to manage the event loop is the following:

```
running = True
while running:
    for event in pygame.event.get():
        # Do something with event
        if event.type == pygame.KEYDOWN:
            # The user pressed a key: the key code is available in event.key
            # You can translate it to a string by using pygame.key.name()
            keycode = event.key
            keyname = pygame.key.name(keycode)
            print(keyname)
```

Examples of events

`event.type`	**User action**	**Attributes**
`MOUSEMOTION`	Moved the mouse	pos, rel, buttons
`KEYDOWN`	Pressed a key	key, mod, unicode, scancode
`MOUSEBUTTONDOWN`	Pressed the mouse button	pos, button
`QUIT`	Clicked the close button	None

See the reference: https://www.pygame.org/docs/ref/event.html

Examples of keys

`event.key`	**Keyboard press**
`K_UP`	Up arrow
`K_DOWN`	Down arrow

`event.key`	**Keyboard press**
`K_k`	Letter k

See the full list: https://www.pygame.org/docs/ref/key.html

Exercise 2.1

Complete the code in `L2_event_loop.py`. The program must display the name of the key that is pressed in the window.

Hints

You will need to draw text. Use `pygame.font`. For example, the following code will return a surface with the text "Hello!" rendered in font Arial, size 18:

```
# Initialize pygame font system
pygame.font.init()
# Get the font from system fonts
arial = pygame.font.SysFont('arial', 18)
# Create the surface
text_surface = arial.render("Hello!", True, (0, 0, 0))
#                                 antialiased   origin of text
```

Note: pygame does not erase graphics when blitting / drawing on top of an existing surface. You may need to repaint it fresh (`surface.fill`) before blitting on it again.

Exercise 2.2

Modify the `L2_event_loop.py` file:

- create a shape of your choice in the top right corner
- make the program exit when the user clicks in the shape
  - you will have to exit from the infinite event polling loop: `running = False`

> The event `pygame.QUIT` is triggered when the user clicks the "Close" button of the application. It is a good idea to always implement it.

Understand FPS and become the master of time

A typical approach in a pygame program is the following:

```
while running:
    for event in pygame.event.get():
        # Do things

    pygame.display.update()
```

This is essentially an infinite loop: Python will run this block of code over and over, as fast as your computer can manage. This also means that pygame will refresh the display as often as your computer can. This will most likely drain resources from your operating system and make your computer very slow. It is common to set the refresh rate of the display to a specific value: the FPS (frames per second).

Use `pygame.time`

Pygame comes with a time management module. It allows you to create a `Clock` object.

```
clock = pygame.time.Clock()
```

You can use this object to set the framerate of your game. Calling `clock.tick(FPS)` will "block" your program and make sure that it never runs at more than `FPS` per second. For many 2D games, a value of 30 or 40 is usually fine.

Exercise 3: 4K @ 120 FPS

Modify the `L3_lag.py` file: change the FPS value and notice how the game behaves.

Create a text surface in the middle of the window, and display the number of milliseconds since the program was started. Use the `get_ticks()` method from Pygame. Notice how often the time is updated, depending on the FPS value.

> Note: this example uses an other way to obtain the pressed keys (outside of the event loop). This is useful when you expect several keys to be pressed at the same time, or combination of keys. Both ways work.

Using images and sprites

Pygame comes with a system of `Sprite`s. A "sprite" is a computer graphic (usually in 2D) that can be used and manipulated as a single entity (instead of a list of pixels). It is very useful and convenient to use sprites. Another reason to use them is because Pygame can detect collisions between sprites - which allows you to detect if two objects overlap on the screen.

Using images in pygame

Pygame can load images as a regular surface. For example:

```
my_surface = pygame.image.load("waldo.png")
```

It can be useful to set the color key for transparency (typically on black pixels).

Other useful functions include:

```
# Will scale `my_surface` to 50x50 pixels
my_scaled_image = pygame.transform.scale(my_surface, (50, 50))

# Will flip `my_surface` on X axis
my_flipped_image = pygame.transform.flip(my_surface, True, False)

# Will rotate `my_surface` 45 degrees clockwise
rotated = pygame.transform.rotate(my_surface, 45)
```

The `Sprite` class

Your custom sprite **MUST** inherit `pygame.sprite.Sprite`, and call its constructor:

```
class MySprite(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
```

Very important attributes

Your custom sprite must define two public attributes:

- `self.image`: contains the image of the sprite (Surface)
- `self.rect`: contains the bounding box of the sprite

You can move the sprite by changing the values `self.rect.x` and `self.rect.y`.

Your custom sprite **SHOULD** define an `update()` method. In this method, you can move the sprite (change `self.rect.x` and `self.rect.y`, or do other operations). It will be useful with sprite groups.

```
class MySprite(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.image.load("waldo.png")
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.y += 5
```

You can then use it in your program:

```
sprite = MySprite()
running = True

while running:
    clock.tick(40)
    sprite.update()
    window.blit(sprite.image, sprite.rect)
```

Putting sprites in sprite groups

Sprites really shine when organized in groups. Instantiate `pygame.sprite.Group` to create a sprite group.

Add sprites by calling the `add` method on the group.

Call the `update` method on the group to call all the `update` methods of the individual sprites in the group. Call the `draw` method on the group to blit all the sprites at their respective locations (defined by `self.rect`).

```
window = pygame.display.set_mode((500, 500))

group = pygame.sprite.Group()

for idx in range(10):
    group.add(MySprite())

while something:
    # Update sprite positions
    group.update()
```

```
        # Blit all sprites
        group.draw(window)
        # Update screen
        pygame.display.update()
```

Collision detection

Pygame can detect collisions between:

- two sprites
- a sprite group and a sprite
- two sprite groups

See the official documentation for more details.

Exercise 4: put it all together

- Look at the `L4_sprites.py` file, and make sure you understand what is going on.
- The program uses sprites, sprite groups, event management, and collision detection.
- What changes would you make to the game?