CSE 1321L: Programming and Problem Solving I Lab

Lab 9

Multimedia

What students will learn:

* Opening and using image files
* Transforming Surfaces
* Playing sounds
* Writing to the screen

Overview: So far, we’ve only used rectangular Surfaces in our labs. However, Pygame features several modules which allow you to easily add rich content to your games, such as pictures from your disk, sounds, and music. We’ll go over some of those in today’s lab, but more can be found in Pygame’s documentation, as well as the lecture slides.

Image Module: Pygame has a module called “image”, which allows you to load images from your hard drive. Note that you will not be creating image objects; instead, the functions will give you normal Surface objects, which you can then blit onto some other Surface.

picture = pygame.image.load(“filename”).convert()

picture = pygame.image.load(“filename”).convert\_alpha()

The code above will create a Surface based on the file “filename” that you pass. Use convert() if your image does not have transparency, and use convert\_alpha() if your image has transparency.

Transform Module: This module contains several different functions to apply transformations to your image, such as rotation, flipping, and scaling. Notice that most of these methods are destructive,meaning that information on the original image will be lost. **When applying a transformation, always save the result onto a different variable; never save it back into the original variable.**

**original = pygame.image.load(“filename.png”).convert()**

flip\_horizontally = True

flip\_vertically = True

**flipped\_image = pygame.transform.flip(original, flip\_horizontally, flip\_vertically)**

new\_width = 50

new\_height = 50

**scaled\_image = pygame.transform.scale(original, (new\_width, new\_height))**

new\_angle = 40

**rotated\_image = pygame.transform.rotate(original, new\_angle)**

**Flip**: Flips the image, either horizontally, vertically, or both. Takes in 3 parameters: the Surface to be flipped, and two Booleans. Returns a Surface.

**Scale**: Resizes the image. Takes in 2 parameters: the Surface to be scaled, and the dimensions of the scaled Surface. Returns a Surface.

**Rotate**: Rotates a Surface. Takes in 2 paramters: the Surface being rotated, and the angle of rotation. Positive numbers rotate counterclockwise, while negative numbers rotate clockwise. Returns a Surface.

Mixer Module: This module allows for the creation of Sound objects, which can then be played. You can create a Sound object as follows:

sound = pygame.mixer.Sound(“filename”)

Once you have a sound object, you can use the following methods:

sound.play() # plays the sound

sound.stop() # stops playing the sound

sound.set\_volume(volume) # sets the volume of the sound

sound.get\_volume() # returns the current volume of the sound

Notice that the sound volume must be a number between 0 and 1 inclusive, with 0 being not making any sound, 1 being the sound at its loudest, and 0.5 being midway between the two.

It’s worth noticing that you can check if the Mixer is currently playing a Sound using the method below, which returns a Boolean:

pygame.mixer.get\_busy()

Font Module: This module allows the creation of Font objects, which can be used to write things to the Display Surface. First, you must create a Font object:

font\_object = pygame.font.Font("font\_file", 32)

The Font constructor takes in 2 parameters: The name of the file holding the font, and the size you want the font to be. Notice that you can pass a None as your first parameter, making Pygame use its default font. Once the Font object has been created, you can create a Surface containing the text you wish to write using render():

smooth\_font = False  
rendered\_font = font\_object.render("text to render”, smooth\_font, (255,255,255))

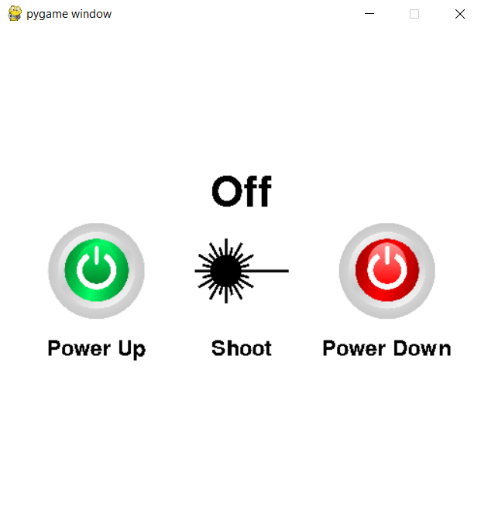
The Render method takes in 3 parameters: a string you want to write, a Boolean indicating whether you want the font to have its edges smoothed out, and the color of the font. It returns a Surface containing your text.

Finally, if you wish to know the area of your font’s Surface, you can use the get\_rect() method. This method returns a Rect, with all of its usual fields:

rendered\_font\_rect = rendered\_font.get\_rect()

This Rect can then be used to place your font’s Surface where you want it to go.

rendered\_font\_rect.center = screen.get\_rect().center  
screen.blit(rendered\_font, rendered\_font\_rect)

**Lab9**: Implement the program in the picture to the right. A video has also been provided showing how this program is supposed to work while in action. Note the following:

* General Info:
  + The screen’s size is 500x500.
* Files:
  + The 3 images needed are provided.
  + The 3 sound files needed are provided.
* Images:
  + All images files are either 300x300 or 400x400.
  + You must scale the images down to 100x100 before using them as buttons.
  + The laser image’s center is at the exact center of the screen.
  + Both the on and the off button are aligned horizontally with the laser button.
  + The on button’s right border is 20 pixels away from the laser button’s left border.
  + The off button’s left border is 20 pixels away from the laser button’s right border.
* Button Font:
  + This refers to all the Font objects rendered under the buttons.
  + All of these Font objects use the default Pygame font, and are size 32.
  + All of these font Surfaces are vertically aligned with their respective buttons.
  + All of these font Surfaces’ top border are 20 pixels down from their respective button’s bottom border.
* Status Font
  + This refers to the Font object right above the laser button.
  + This Font object uses the default Pygame font, and is size 64.
  + This font’s Surface is vertically aligned with the center of the screen.
  + This font’s Surface center is 80 pixels above the center of the screen.
* Sounds:
  + Only one sound can be playing at a time.
* Functionality:
  + While the laser is “Off”, the Shoot and Power Down button do nothing.
  + Upon pressing the Power Up button, play the power up sound and change the status to “Charging”.
  + When the power up sound finishes playing, change the status to “Ready”.
  + While the laser is “Ready”, the Power Up button does nothing.
  + While the laser is “Ready”, pressing the Shoot button plays the shoot sound and changes the status to “Shooting”.
  + When the shooting sound is done playing, change the status back to “Ready”.
  + While the laser is “Ready”, pressing the Power Down button plays the power down sound and changes the status to “Discharging”.
  + When the power down sound finishes playing, change the status to “Off”.
  + While the laser is “Charging”, “Discharging”, or “Shooting”, none of the buttons do anything.

Hint: You should use a couple of extra Boolean variables to keep track of when the laser is charged and when the laser is busy doing something.

**Submit your file to Gradescope as Lab9.py**.