pyGame Lecture #8

(Examples: soundbutton, badmario)

SOUND IN PYGAME

I. Power of Sound

Games are supposed to fully immerse a user into some virtual reality. Incorporating sound into a game greatly enhances the users experience and can have a large impact on brining your game to life. There are two main forms of sounds in video games: noises that result from players actions and music that plays in the background. PyGame has two options: music and sounds. Music will play in the background of your game while a sound will play when you call them.

II. Sound modules

The pyGame mixer module contains the classes you will need to implement sounds in your game.

For a full reference of sound events, go to:
https://www.pygame.org/docs/ref/mixer.html

The pyGame music module if very close to the mixer module, but the music module will allow you to implement background music in your game.

For a full reference of music events, go to:
https://www.pygame.org/docs/ref/music.html

III. Soundbutton example

In this example we'll use the pyGame mixer module to demonstrate how to incorporate sounds into your game. For this program we'll just have a simple button that generates a random sound when clicked.

Step 1: Assign sound file

The pyGame mixer module has a Sound function that creates a new Sound object from a file. To use this just pass the filename of your sound in the parameter.

This function is called in the soundbutton example as follows:

```
blip_sound = pygame.mixer.Sound("blip.wav")
boing_sound = pygame.mixer.Sound("boing.wav")
bubbles_sound = pygame.mixer.Sound("bubbles.wav")
```

Step 2: Play sound

To play a sound in pyGame you can call the play function to begin the playback of the sound.

This function is called in the soundbutton example as follows:

```
pygame.mixer.Sound.play(sounds[idx])
```

The whole program, with comments is included on the following pages.

```
# Ellie Kozlowski
# Python Sound Example
import pygame
import random
import sys
pygame.init()
display = pygame.display.set_mode((650, 350))
white = (255,255,255)
display.fill(white)
button = pygame.image.load("button.png")
button = pygame.transform.scale(button, (200, 200)) # rescale button
display.blit(button, (220, 100))
myfont = pygame.font.SysFont("monospace", 30)
label = myfont.render("Press the button to make a sound!", 1, (0, 0,
0))
display.blit(label, (20, 50))
```

```
pygame.display.update()
# assign sound files
blip_sound = pygame.mixer.Sound("blip.wav")
boing_sound = pygame.mixer.Sound("boing.wav")
bubbles_sound = pygame.mixer.Sound("bubbles.wav")
# list to hold sound files
sounds = [blip_sound, boing_sound, bubbles_sound]
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.quit()
        # check if button is pressed
        if event.type == pygame.MOUSEBUTTONDOWN:
            if pygame.mouse.get_pressed()[0]:
                # generate random idx in list
                idx = random.randint(0, 2)
                # play sound
                pygame.mixer.Sound.play(sounds[idx])
```

IV. Badmario example

In this example we'll use the pyGame music module to play the Mario theme music in the background of this simple Mario game.

Step 1: Load music

To load a music file for playback pyGame has a load function that is passed the file name of your music.

This function is called in the badmario example as follows:

```
pygame.mixer.music.load("theme.wav")
```

Step 2: Play music

To start the playback of a music stream pyGame has a play function that can be passed multiple parameters. The first parameter is the loops argument which specifies the number of times a song will repeat. For example, play(4) will make the song play once, then four more times, for a total of five. If you pass a -1 then the song will repeat indefinitely. The second parameter is the starting position argument which controls where the song starts playing. The starting position is dependent on the format of music playing.

This function is called in the badmario example as follows:

```
pygame.mixer.music.play(-1)
```

A -1 is passed in the parameter so that the music will repeat indefinitely.

The whole program, with comments is included on the following pages.

```
#Bryan Medina
#6/27/17
#badmario.py
#Crappy Mario....
import pygame, sys
from math import *
from random import randint
from pygame.locals import *
pygame.init()
DISPLAYSURF = pygame.display.set_mode((1000, 600))
mario = pygame.image.load("mario.png")
mario = pygame.transform.scale(mario, (64, 64)) #Resizing Mario image
background = pygame.image.load("background.jpg")
background = pygame.transform.scale(background, (1000, 600)) #Resizing
background image
shell = pygame.image.load("shell.png")
shell = pygame.transform.scale(shell, (32, 32)) #Resizing shell
```

```
pygame.mixer.music.load("theme.wav") #loads the music
pygame.mixer.music.play(-1) #play music infinitely
shells = [] #List for all the shell
#variables to keep track of the width and height of the screen
width  = 1000
height = 600
velocity     = 0 #Mario's Y velocity
velX         = 0 #Mario's X velocity
posX         = 500 #x position of our ball
posY         = 530 #y position of our ball
GROUND       = 500 #the surface our ball is sitting on
gravity      = 2 #The reason why we always fall back to the ground
while True: #Game loop
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.quit()
        if event.type == KEYDOWN:
            if event.key == K_w: #if w is pressed
                velocity = -30 #inital y velocity will be non zero
            if event.key == K_a: #if a is pressed
                velX = -10 #Mario will move to the left
            if event.key == K_d: #if d is pressed
                velX = 10 #Mario will move to the right
            if event.key == K_SPACE: #if space is pressed
                shells.append([posX+60, posY, 20, shell])
                  #Shell will be thrown
                  #WARNING: this list will get huge fast... Make sure to
                  remove offscreen shells
        if event.type == KEYUP: #if the key is released
            velX = 0 #Mario's X velocity will be 0
    for shelly in shells: #Update the position of every shell in the list
```

```
    shelly[0] += shelly[2]
velocity += gravity #add gravity to velocity
posY += velocity # add velocity to position
posX += velX #Change position of mario
if posX >= width: # make him wrap around the screen
    posX = 0
elif posX <= 0:
    posX = width
if posY + 20 >= GROUND and velocity > 0:
  #This keeps Mario above (or on theground)
    velocity = 0
    posY = GROUND
DISPLAYSURF.blit(background, (0, 0))  #Display background
DISPLAYSURF.blit(mario, (posX, posY)) #Display Mario
for shelly in shells: #Display all shells
    DISPLAYSURF.blit(shelly[3], (shelly[0], shelly[1]))
pygame.display.update()
```