Attendance

Introduction to Pygame

Presented by:

ACM at University of Southern Indiana

Jakobie Brown

Requirements

Python

- Download python from
- https://www.python.org/downloads/
- Run the installer:
  - Windows: Ensure "Add Python to Path" is checked when installing

A plain text editor and a shell environment

-or-

An Integrated Development Environment (IDE)

- VS Code
- Etc..

Set Up

Download or clone repository from

https://www.github.com/JakobieBrown/PygameIntro

(link in discord)

Open the directory in a terminal or IDE

Install pygame.

In your terminal type:

```
pip install pygame
```

Initializing Pygame

import pygame

- Import the pygame module

pygame.init()

- Required before using pygame

pygame.display.set_mode((WIDTH,HEIGHT))

- Creates a window with a pygame.Surface of the desired size

while True:

- Loop forever

Handling Events

Without handling events, the program is unresponsive.

- Windows: press ‘ctrl’ + ‘c’ in the console to kill the program
- Mac: press ‘⌘’ + ‘.’

Get a List of the events in the main loop:

```
events = pygame.event.get()
```

Iterate through the events:

```
for event in events:
    if event.type == pygame.<event type>:
```

Quitting Pygame

pygame.quit()

- The opposite of pygame.init()
- quit() safely uninitializes all pygame modules
- After calling quit(), pygame will no longer work

Quit Event

- The quit event is triggered when the user presses close on the window.
- Check for this event in the event loop
- if event.type == pygame.QUIT:

```
  pygame.quit() # quit pygame
  exit() # exit program
```

Setting FPS

pygame.time.Clock

Used to manage timing and the frame rate

Create a clock instance

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

In the main loop:

```
    clock.tick(30) # sets the framerate to 30 fps
```

Rendering Images

Pygame renders images with pygame.Surface.

Surfaces are drawn on the main surface with

main_surface.blit()

In the main loop call pygame.display.update() to re-render the surfaces

Pygame axis begins at the top left

+x is to the right

+y is downward

Pygame Surface

pygame.Surface

- Used to draw images on the pygame window

Four “types” of surfaces

- Basic surface
- Main surface
    - The base surface all other surfaces are drawn on top of
- Image surface
    - Used to display image data
- Text surface
    - Used to render text

Surface.blit(surface, rect)

- Used to draw any surface to any other surface

Pygame Surface

Main Surface

- screen = pygame.display.set_mode((width,height))
- Named screen by convention

Basic Surface

- surface = pygame.Surface((width,height))
- surface.fill(color) #fill surface with a color

Image Surface

- image = pygame.image.load(path/to/image)
- .convert_alpha() # converts the image to a usable format and preserves the alpha value.

Pygame Surface

Text Surface

- Rendered from a Font object
- text_surface = font.render(txt, antialias, color, bg_color)

Font

- font = pygame.font.Font(path/to/font | None, size)
- font = pygame.font.SysFont("FontName", size, bold, italic)

Example (drawing a surface to the main surface)

- box = pygame.Surface((32,32)) #create 32x32 square
- box.fill("blue") #fill square with blue
- screen.blit(box,(x,y,w,h))#draw surface to screen

Pygame Rect

Rectangles in pygame are very useful

They’re used to:

- Accurately place and move surfaces
- Detect collisions

```
image = pygame.image.load(path).convert_alpha()
Image_rect = image.get_rect() # create rect from image
screen.blit(image,image_rect) # blit image to screen
```

Pygame Rect

Using rects enables us to place images accurately without using geometry

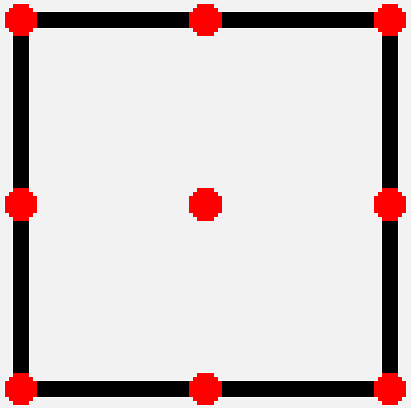
```
rect = surface.get_rect(point* = (x,y))
```

This enables you to set a point of the rect to a specific coordinate values.

topleft midtop topright

midleft center midright

bottomleft midbottom bottomright



Example (draw text in top right corner)

```
txt = font.render(txt,True,’white’, None)

rect = txt.get_rect(topright = (screen_width, 0)

screen.blit(txt,rect)
```

Animation (movement)

Motion in videos is caused by changing the position of an image over time.

The rects we set up for our images will make this very simple.

Example:

```
image = pygame.image.load(path)
rect = image.get_rect()
speed = c
while True:
    rect.x += speed
    screen.blit(image, rect)
```

Animation (movement): 2-Dimensional

For 2-dimensional movement:

- Deal with components separately
- Cos is for x
- Sin is for y

Example:

```
direction = (direction + delta_dir) % 360
velocity.x += cos(direction)
velocity.y += sin(direction)
position.x += velocity.x
position.y -= velocity.y # inverted y-axis
```

Collision Detection

Rects enable us to check for collisions between two rects

rect.colliderect(other_rect)

- Returns Boolean value to indicate if rects overlap
- Problem: Calling this for all possible colliding rects would be tedious and/or inefficient

Key Events

Pygame provides an interface that triggers events when keys are pressed

To respond to these events we listen for them in the event loop.

```
for event in pygame.event.get():
        if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_<key>:
                        #do something
        elif event.type == pygame.KEYUP:
                if event.key == pygame.K_<key>:
                        #do something
```

Surface Transformation: Rotation

pygame.transform.rotate(surface, angle)

- Returns the transformed surface
- Angle is in degrees
- Keep an unaltered surface to transform to avoid issues
- Reset center after rotation
- +angle: counterclockwise
- -angle: clockwise

Example

```
image = pygames.image.load(path).convert_a...
rect = image.get_rect(center = (x,y))
rotated = pygame.tansform.rotate(image, angle)
rect = rotated.get_rect(center = rect.center)
```

Surface Transformation: Scale

pygame.transform.scale(surface, 2d size)

- Scales surface to the size provided
- Size is two values (w,h)

pygame.transform.scale2x(surface)

- Scales the surface by a factor of two

pygame.transform.scale_by(surface, factor)

- Scales the surface by the factor provided

Same tips for rotation apply to scale

Example:

```
image = pygames.image.load(path).convert_a…
rect = image.get_rect(center = (x,y))
scaled = pygame.tansform.scale(image, size)
rect = scaled.get_rect(center = rect.center)
```

Physics based movement: Acceleration & Velocity

So far, we have moved our surfaces by a constant amount

To emulate kinematic motion, we can move our surface by variable amount

To accomplish this, we'll use two values. Acceleration and velocity.

In the main loop:

```
        acceleration += c  #constant
        velocity += acceleration
        position += velocity
```

This has the effect of "gaining speed" over time

It works great for a frictionless environment

Physics based movement: Drag

With the acceleration and velocity working together to move the surface, you've probably noticed it becomes too fast

To fix this we need to create a terminal velocity

Terminal velocity occurs when the fricative force becomes large enough to oppose the accelerating force, making acceleration drop to zero and causing the velocity to become constant

Example:

```
friction_coeff = .04
acceleration += c
velocity += acelleration
friction = friction_coeff*velocity
velocity -= friction
```

As the velocity grows, so does the friction eventually reaching terminal velocity

Basic Game States

Creating a basic game state is simple

We change what happens in the main loop

Example:

```
active = boolean
while True:
    if active:
        # play game
    else:
        # show start screen
```

Mouse Events

Pygame provides an interface that triggers events from the mouse

- event.pos # position of mouse in a mouse event

Mouse Events

- pygame.MOUSEDOWN #when mouse button is pressed
- pygame.MOUSEUP #when mouse button is released
    - event.button # represents which button was pressed/released
        - Left-click: 1 | Wheel-click: 2 | Right-click: 3
        - Wheel-scroll up: 4 | Wheel-scroll down: 5
- Pygame.MOUSEMOTION
    - event.rel # change in position
- pygame.WHEEL
    - Event.y # how much the wheel scrolled vertically
    - Event.x # how much the wheel scrolled horizontally

Shapes and lines

Pygame provides an interface for drawing shapes and lines on surface

Example

pygame.draw.line(surface, color, start, end, width)

Example

rect = (x,y,w,h)

pygame.draw.ellipse(surface,color,rect)

Colors

In graphic art we use additive colors Red, Blue, and Green

- Any color can be made by combining red, blue, and green at varying values
- Red, Blue, and Green can be values in range [0-255]
- Numbers can be represented with hexadecimal values [000000 – FFFFFFF]

Pygame handles colors in 3 different ways

- By a 3 integer tuple: (R,G,B)
- By a hexadecimal string: ‘#RRGGBB’
- By the name of the color: ‘color_name’

Colors

Examples

```
red = pygame.Color((255,0,0))
blue = pygame.Color(‘#0000FF’)
green = pygame.Color(‘green’)
```

Alpha Value

Colors also have an Alpha value in range [0-255] that determines the transparency of the color

Example

```
transparent_red = pygame.Color((255,0,0,128))
transparent_blue = pygame.Color(‘#0000FF80’)
```

Sounds

Load sound:

```
sound = pygame.mixer.load(‘path/to/sound’)
```

Adjust volume:

```
sound.set_volume(percentage)
```

Play Sound once:

```
sound.play()
```

Play Sound multplie times:

```
sound.play(n) #plays sound n times
```

Loop forever:

```
sound.play(-1)
```

Stop:

```
sound.stop()
```

Sprite class

The sprite class is used to unify the surface and the rect under one object

Example

```
class Player(pygame.sprite.Sprite):
        self.__init__(self):
                super().__init__()
                self.image = pygame.Surface((w,h))
                self.rect = self.image.get_rect()
```

self.image and self.rect are required attributes in a pygame Sprite

This may seem no different than if you were to create a class, but the real advantage is when you put sprites into groups.

Sprite Groups

A group in pygame is a collection of sprites

Pygame has two types of sprite groups

Group for multiple sprites

GroupSingle for a single sprite

Group has a draw method that uses a sprites, image and rect to draw the sprites on a surface

Group has an update method that calls the update method of sprites in the group

Creating groups:

player_group = pygame.sprite.GroupSingle()

Projectile_group = pygame.sprite.Group()

Sprite Groups

Drawing sprites in a group to screen:

```
    player_group.draw(screen)
```

Updating sprites:

```
class Player(pygame.sprite.Sprite):
    self.__init__(self):...
    def update(self):
        #update player
player_group.update()
```

Referencing Sprites in groups:

```
player = player_group.sprite # for GroupSingle
projectile = projectile_group.sprites # List, for Group
```

Spawning Objects

Now that we have our groups and sprites drawing to the screen

We can spawn objects simply by adding them to their group.

```
player_group.add(Player())
```

Group Collision

One of the largest benefits of Sprite Groups is to efficiently handle collisions

To check for collisions between two sprite groups:

GroupSingle:

pygame.sprite.spritecollide(group.sprite, other_group, bool)

- If bool is true, the sprite from other_group is destroyed
- Returns a list of sprites in other_group detected in the collision

Group:

pygame.sprite.groupcollide(groupA, groupB, destroyA, destroyB)

- Returns a Dict of <spriteA, List<sprite>>

Group Collision

Examples:

Single collide with group

```
for sprite  in pygame.sprite.spritecollide(a.sprite, b, False,):
        print(f’{a.sprite} collided with {sprite}”)
```

Group collide with group

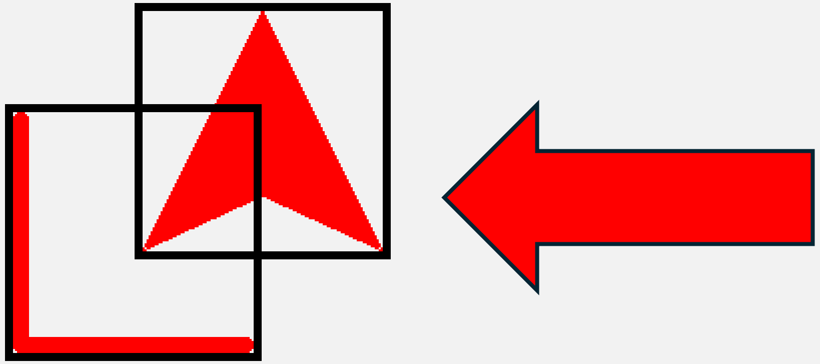
```
for sprite_a, list_b in pygame.sprite.groupcollide(a,b,False,False).items():
        for sprite_b in list_b:
                print(f’{sprite_a} collided with {sprite_b}’)
```

Pixel-perfect Collisions

At this point you may have noticed the collisions look a bit strange.

This is caused by how colliderect() behaves

colliderect checks if the rects are overlapping and returns true if they are



Collision!

Pixel-perfect Collisions

To fix this, we’ll use masks.

Creating a mask:

image = pygame.image.load(path)…

mask = pygame.mask.from_surface(image)

Comparing masks:

a.mask.overlap(b.mask, offset) -> Boolean

The offset is the difference between the topleft coordinates of the two sprites

To correctly calculate this:

```
offset = (b.rect.left – a.rect.left,
          b.rect.top – a.rect.top)
```

Introduction to Pygame

Presented by:

ACM at University of Southern Indiana

Jakobie Brown