Hands-on Coding Session I
DevGame by PyGame

Dr. Srisupang Thewsuwan

Pygame

- PyGame is an open-source library for making video games in Python.
- The main features, including 2D graphics, sound, user input, and collision detection.
- PyGame is cross-platform and works on various operating systems.

What is pyGame?

- A set of Python modules to make it easier to write games.
  - home page: http://pygame.org/
  - documentation: http://pygame.org/docs/ref/

- pyGame helps you do the following and more:
  - Sophisticated 2-D graphics drawing functions
  - Deal with media (images, sound F/X, music) nicely
  - Respond to user input (keyboard, joystick, mouse)
  - Built-in classes to represent common game objects

pyGame at a glance

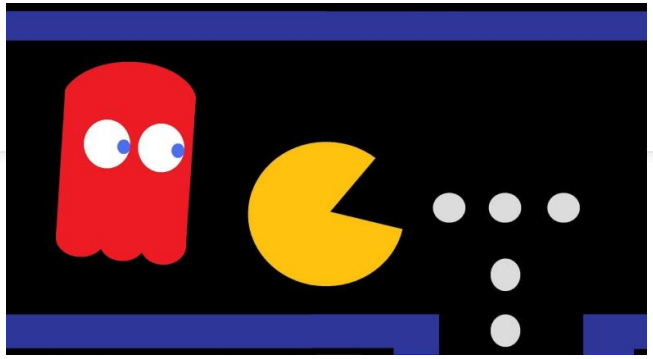
- pyGame consists of many modules of code to help you:

```
cdrom       cursors       display       draw      event
font        image         joystick      key       mouse
movie       sndarray      surfarray     time      transform
```

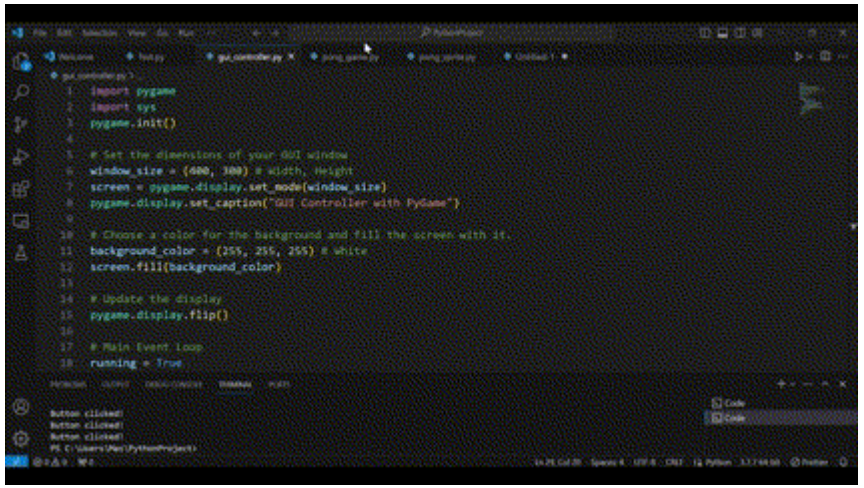
- To use a given module, import it. For example:

```
import pygame
from pygame import *
from pygame.display import *
```

Game fundamentals



- sprites: Onscreen characters or other moving objects.
- collision detection: Seeing which pairs of sprites touch.
- event: An in-game action such as a mouse or key press.
- event loop: Many games have an overall loop that:
  - waits for events to occur, updates sprites, redraws screen



File Edit Selection View Go Run ··· PythonProject

Welcome | Test.py | gui_controller.py | pong_game.py | pong_sprite.py | Untitled-1

gui_controller.py > ...

```python
import pygame
import sys
pygame.init()

# Set the dimensions of your GUI window
window_size = (400, 300) # Width, Height
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("GUI Controller with PyGame")

# Choose a color for the background and fill the screen with it.
background_color = (255, 255, 255) # White
screen.fill(background_color)

# Update the display
pygame.display.flip()

# Main Event Loop
running = True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Button clicked!
Button clicked!
Button clicked!
PS C:\Users\Mas\PythonProject>
```

Code
Code

0 0 0 Ln 29, Col 20 Spaces: 4 UTF-8 CRLF Python 3.7.7 64-bit Prettier

A basic skeleton components

- Import necessary libraries
- Initialize PyGame
- Define game window settings
- Create the game loop
- Handle events
- Update game objects
- Render game objects
- Clean up and quit

```
import pygame
import sys
pygame.init()
```

Import and Initial Pygame

```
# Set the dimensions of your GUI window
window_size = (400, 300) # Width, Height
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("GUI Controller with PyGame")

# Choose a color for the background and fill the screen with it.
background_color = (255, 255, 255) # White
screen.fill(background_color)

# Update the display
pygame.display.flip()

# Main Event Loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            if button_rect.collidepoint(event.pos):
                print("Button clicked!")
    button_color = (0, 255, 0) # Green
    button_rect = pygame.Rect(150, 125, 100, 50) # x, y, width, height
    pygame.draw.rect(screen, button_color, button_rect)

    # Update the display
    pygame.display.flip()

# Clean up and quite
pygame.quit()
sys.exit()
```

A basic skeleton components

- Import necessary libraries
- Initialize PyGame
- Define game window settings
- Create the game loop
- Handle events
- Update game objects
- Render game objects
- Clean up and quit

```
import pygame
import sys
pygame.init()

# Set the dimensions of your GUI window
window_size = (400, 300) # Width, Height
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("GUI Controller with PyGame")

# Choose a color for the background and fill the screen with it.
background_color = (255, 255, 255) # White
screen.fill(background_color)

# Update the display
pygame.display.flip()
```

Game window setting

```
# Main Event Loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            if button_rect.collidepoint(event.pos):
                print("Button clicked!")
    button_color = (0, 255, 0) # Green
    button_rect = pygame.Rect(150, 125, 100, 50) # x, y, width, height
    pygame.draw.rect(screen, button_color, button_rect)

    # Update the display
    pygame.display.flip()

# Clean up and quite
pygame.quit()
sys.exit()
```

A basic skeleton components

- Import necessary libraries
- Initialize PyGame
- Define game window settings
- Create the game loop
- Handle events
- Update game objects
- Render game objects
- Clean up and quit

```
import pygame
import sys
pygame.init()

# Set the dimensions of your GUI window
window_size = (400, 300) # Width, Height
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("GUI Controller with PyGame")

# Choose a color for the background and fill the screen with it.
background_color = (255, 255, 255) # White
screen.fill(background_color)

# Update the display
pygame.display.flip()
```

Game loop

```
# Main Event Loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            if button_rect.collidepoint(event.pos):
                print("Button clicked!")
    button_color = (0, 255, 0) # Green
    button_rect = pygame.Rect(150, 125, 100, 50) # x, y, width, height
    pygame.draw.rect(screen, button_color, button_rect)

    # Update the display
    pygame.display.flip()
```

```
# Clean up and quite
pygame.quit()
sys.exit()
```

A basic skeleton components

- Import necessary libraries
- Initialize PyGame
- Define game window settings
- Create the game loop
- Handle events
- Update game objects
- Render game objects
- Clean up and quit

```
import pygame
import sys
pygame.init()

# Set the dimensions of your GUI window
window_size = (400, 300) # Width, Height
screen = pygame.display.set_mode(window_size)
pygame.display.set_caption("GUI Controller with PyGame")

# Choose a color for the background and fill the screen with it.
background_color = (255, 255, 255) # White
screen.fill(background_color)

# Update the display
pygame.display.flip()

# Main Event Loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        if event.type == pygame.MOUSEBUTTONDOWN:
            if button_rect.collidepoint(event.pos):
                print("Button clicked!")
    button_color = (0, 255, 0) # Green
    button_rect = pygame.Rect(150, 125, 100, 50) # x, y, width, height
    pygame.draw.rect(screen, button_color, button_rect)

    # Update the display
    pygame.display.flip()

# Clean up and quite
pygame.quit()
sys.exit()
```

Quit

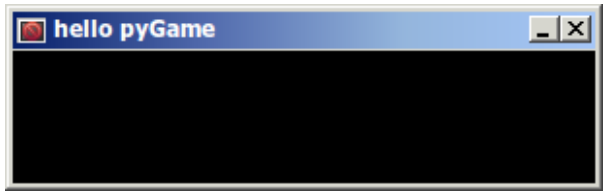
Task#1 Define Game Window Setting

01_pygame_skeleton_structure.py

Initializing pyGame

- To start off our game, we must pop up a graphical window.
- Calling `display.set_mode` creates a window.
  - The call returns an object of type `Surface`, which we will call `screen`. We can call methods on the screen later.
  - Calling `display.set_caption` sets the window's title.

```
from pygame import *

pygame.init()    # starts up pyGame
screen = display.set_mode((width, height))
display.set_caption("title")
...
pygame.quit()
```



Surfaces

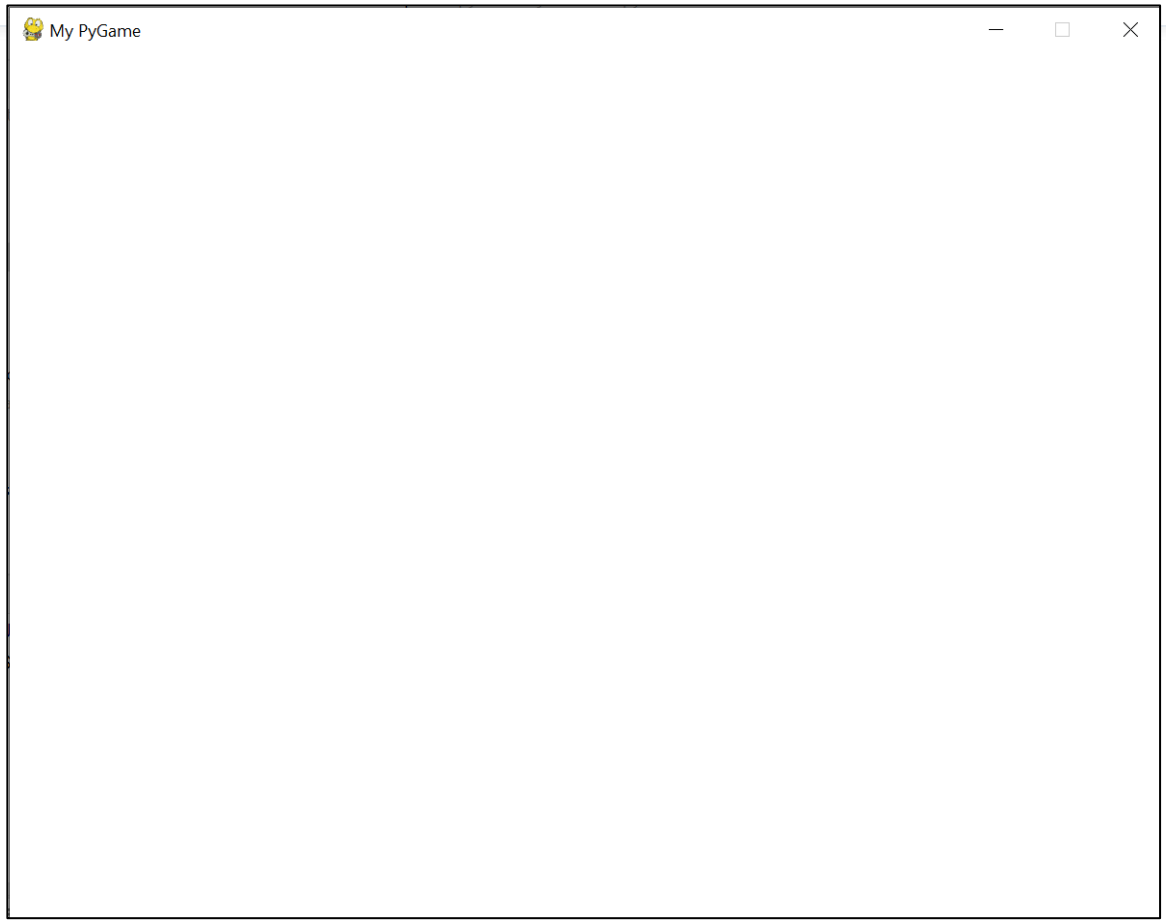
```
screen = display.set_mode((width, height))    # a surface
```

- In Pygame, every 2D object is an object of type `Surface`
  - The screen object, each game character, images, etc.
  - Useful methods in each `Surface` object:

`Surface((`**width, height**`))`	constructs new Surface of given size
`fill((`**red, green, blue**`))`	paints surface in given color *(rgb 0-255)*
`get_width(), get_height()`	returns the dimensions of the surface
`get_rect()`	returns a `Rect` object representing the x/y/w/h bounding this surface
`blit(`**surface, coords**`)`	draws another surface onto this surface at the given coordinates

  - after changing any surfaces, must call `display.update()`

A basic skeleton components

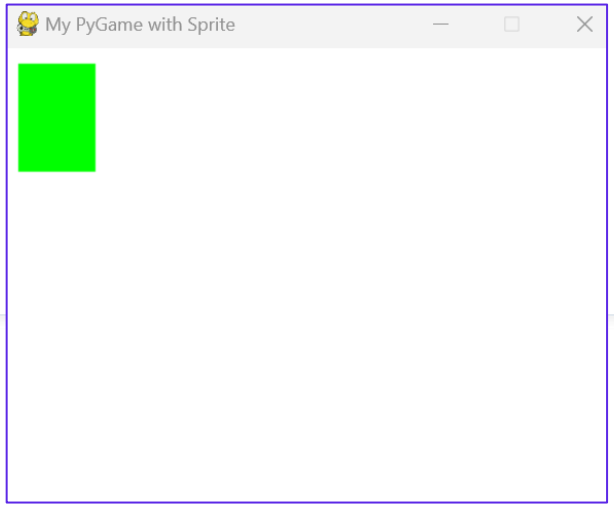



```python
# Import necessary libraries
import pygame
import sys

# Initialize PyGame
pygame.init()

# Define game window settings
WIDTH, HEIGHT = 800, 600
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('My PyGame')

# Define colors
WHITE = (255, 255, 255)

# Game loop
running = True
while running:
    # Handle events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Update game objects
    # (Add code to update game objects' states)

    # Render game objects
    screen.fill(WHITE)
    # (Add code to draw game objects on the screen)

    pygame.display.flip()

# Clean up and quit
pygame.quit()
sys.exit()
```

Sprites

02_pygame_sprites.py

Sprites



- Sprites: Onscreen characters or other moving objects.

- A sprite has data/behavior such as:
  - its position and size on the screen
  - an image or shape for its appearance
  - the ability to collide with other sprites
  - whether it is alive or on-screen right now
  - might be part of certain "groups" (enemies, food, ...)

- In pyGame, each type of sprite is represented as a subclass of the class `pygame.sprite.Sprite`

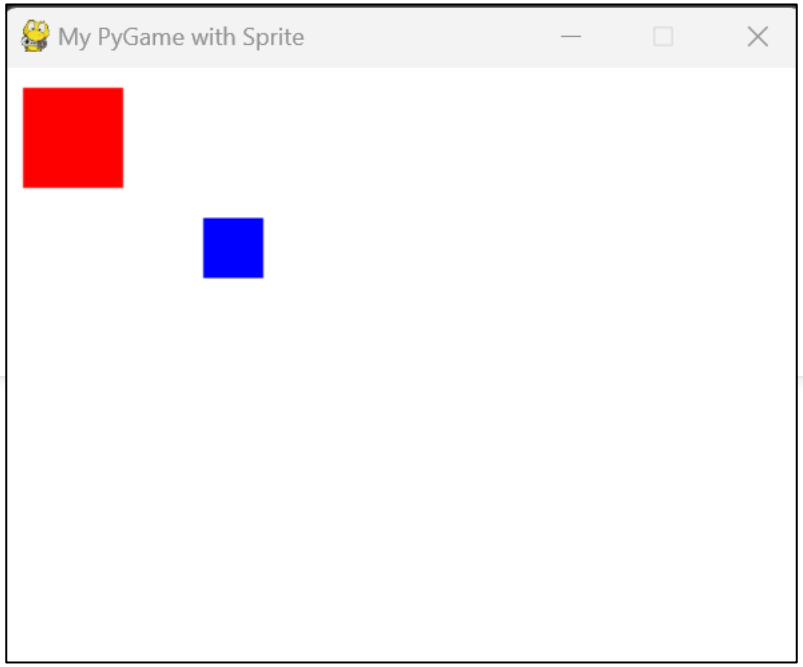
A rectangular sprite

```
from pygame import *
from pygame.sprite import *

class name(pygame.sprite.Sprite):
   def __init__(self):              # constructor
      super().__init__()
      self.image = pygame.Surface(width, height)
       self.image.fill((0,255,0))  # filling sprite with green
       self.rect = self.image.get_rect(leftX, topY, width, height)

   other methods (if any)
```

- Important fields in every sprite:
  - `image` - the image or shape to draw for this sprite (a `Surface`)
    - as with screen, you can `fill` this or draw things onto it
  - `rect` - position and size of where to draw the sprite (a `Rect`)
- Important methods: `update`, `kill`, `alive`

Task#2: Adding a second sprite

Instantiate the Second Sprite



- Create an instance of the sprite class for your second sprite.
- Add this second sprite instance to the same sprite group to manage it alongside your first sprite.

```
 Sprite Class
class MySprite(pygame.sprite.Sprite):
   def __init__(self,x, y, width, height, color):            # constructor
      super().__init__()
      self.image = pygame.Surface((width, height))
      self.image.fill((color))  # filling sprite with green
      self.rect = self.image.get_rect(topleft=(x, y))

# Create sprite instance and sprite group
sprite1 = MySprite(10, 10, 50, 50, (255, 0, 0))  # Red sprite
sprite2 = MySprite(100, 75, 30, 30, (0, 0, 255))  # Blue sprite, different position and size
sprites = pygame.sprite.Group()
sprites.add(sprite1, sprite2)
```

Rect methods

`clip(`**rect**`)` *	crops this rect's size to bounds of given rect
`collidepoint(`**p**`)`	`True` if this `Rect` contains the point
`colliderect(`**rect**`)`	`True` if this `Rect` touches the rect
`collidelist(`**list**`)`	`True` if this `Rect` touches any rect in the list
`collidelistall(`**list**`)`	`True` if this `Rect` touches all rects in the list
`contains(`**rect**`)`	`True` if this `Rect` completely contains the rect
`copy()`	returns a copy of this rectangle
`inflate(`**dx, dy**`)` *	grows size of rectangle by given offsets
`move(`**dx, dy**`)` *	shifts position of rectangle by given offsets
`union(`**rect**`)` *	smallest rectangle that contains this and rect

* Many methods, rather than mutating, return a new rect.

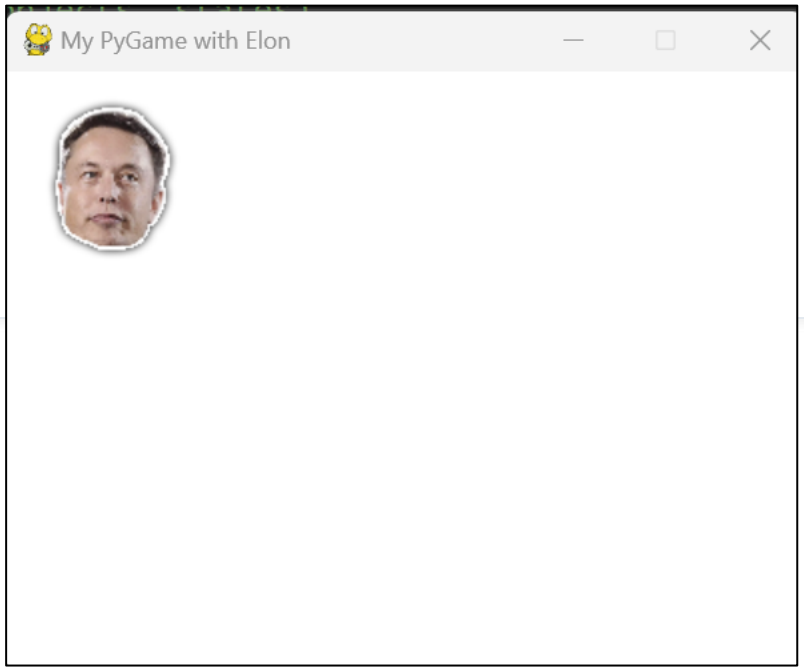
- To mutate, use `_ip` (in place) version, e.g. `move_ip`

A Sprite using an image

```
from pygame import *
from pygame.sprite import *

class name(Sprite):
   def __init__(self):                # constructor
      Sprite.__init__(self)
      self.image = image.load("filename").convert()
      self.rect = self.image.get_rect().move(x, y)

   other methods (if any)
```

- When using an image, you load it from a file with `image.load` and then use its size to define the `rect` field
- Any time you want a sprite to move on the screen,
  you must change the state of its `rect` field.

Setting up sprites



- When creating a game, we think about the sprites.
  - What sprites are there on the screen?
  - What data/behavior should each one keep track of?
  - Are any sprites similar? (If so, maybe they share a class.)

- For our Elon-car game:

```
class Player(Sprite):
    ...
```

Sprite groups

name = `Group`(sprite1, sprite2, ...)

- To draw sprites on screen, put them into a `Group`
- Useful methods of each `Group` object:

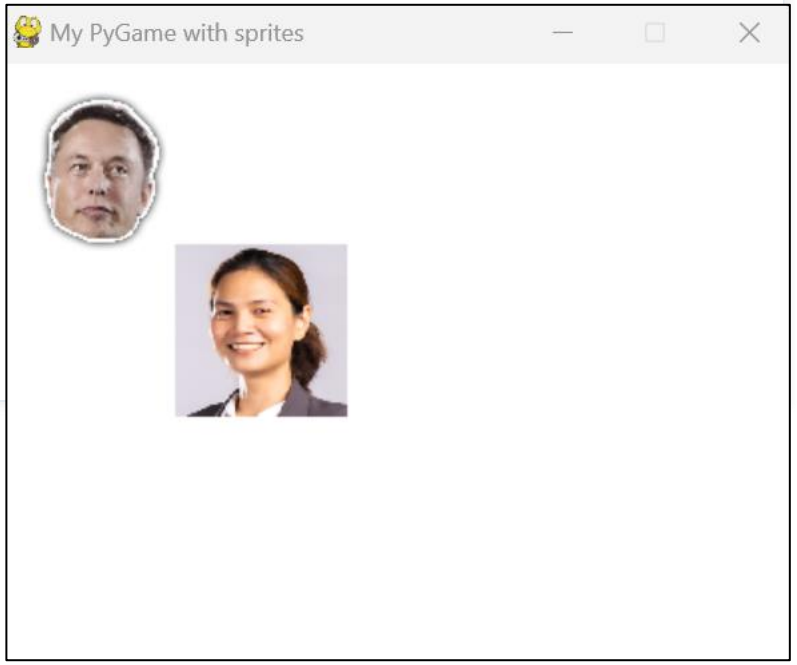
  `draw`(surface) - draws all sprites in group on a `Surface`

  `update()` - calls every sprite's `update` method

```
player1 = Player()     # create  Elon object
player2 = Player()
all_sprites = Group(player1, player2)
...
# in the event loop
all_sprites.draw(screen)
```

Task#3: Create another sprite image (mas.png)

03_pygame_sprites_images.py



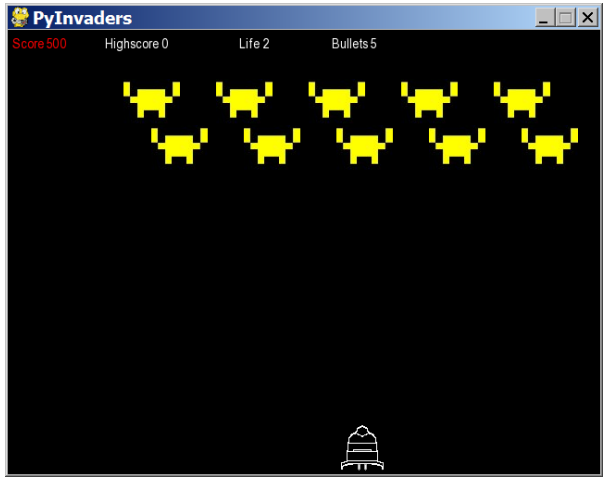

```
# Sprite Class
class MySprite(pygame.sprite.Sprite):
   def __init__(self,x, y,imagepath):            # constructor
      super().__init__()
      original_image = pygame.image.load(imagepath).convert_alpha()
      new_width, new_height = 86, 86  # The new size you want for the image
      self.image = pygame.transform.scale(original_image, (new_width, new_height))
      self.rect = self.image.get_rect(topleft=(x, y))

# Create sprite instance and sprite group
sprite_elon = MySprite(10, 10,'musk.png')  # elon sprite
sprite_mas = MySprite(90, 90,'mas.png')  # mas sprite
sprites = pygame.sprite.Group()
sprites.add(sprite_elon,sprite_mas)
```

Events

Events

- event-driven programming: When the overall program is a series of responses to user actions, or "events."

- event loop (aka "main loop", "animation loop"):
  Many games have an overall loop to do the following:
  - wait for an event to occur, or
    wait a certain interval of time
  - update all game objects (location, etc.)
  - redraw the screen
  - repeat



The event loop

- In an *event loop*, you wait for something to happen, and then depending on the kind of event, you process it:

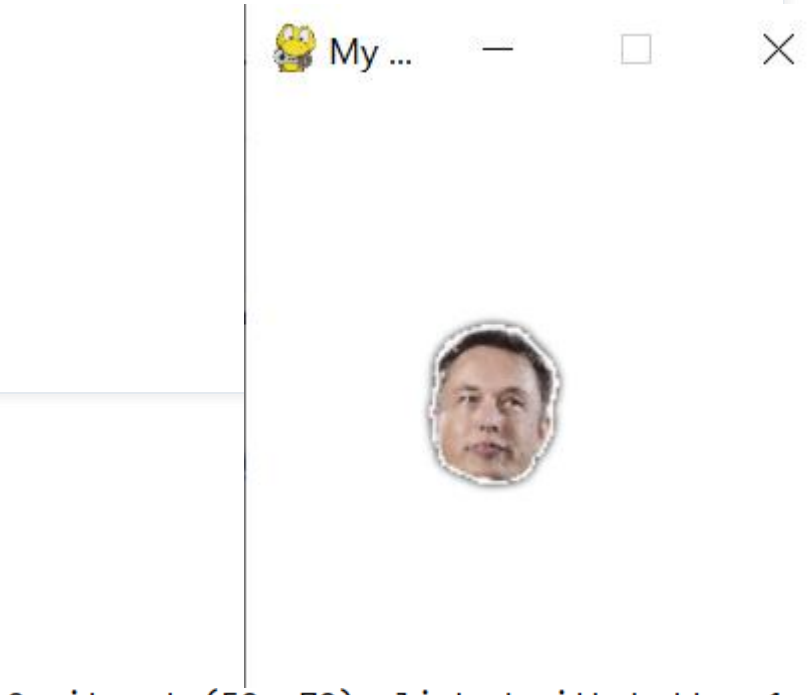
```
while True:
    e = event.wait()        # wait for an event
    if e.type == QUIT:
        pygame.quit()       # exit the game
        break
    elif e.type == type:
        code to handle some other type of events;
    elif …
```

Mouse events

- Mouse actions lead to events with specific types:
  - press button down: `MOUSEBUTTONDOWN`
  - release button: `MOUSEBUTTONUP`
  - move the cursor: `MOUSEMOTION`

- At any point you can call `mouse.get_pos()` which returns the mouse's current position as an (x, y) tuple.

```
e = event.wait()
if e.type == MOUSEMOTION:
    pt = mouse.get_pos()
    x, y = pt
    ...
```




```
Sprite at (50, 70) clicked with button 1
Sprite at (50, 70) clicked with button 1
Sprite at (50, 70) clicked with button 1
```

If the sprite is clicked, it will print a message.

[04_pygame_sprites_events.py]

Go Run ··· PythonProject

pong_sprite.py | 04_pygame_sprites_events.py | pygame-sprites.py | pygame-ClickableSprites.py | pygame-renderText.py

04_pygame_sprites_events.py > ...

```python
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

        ### Add more events here
        if event.type == pygame.MOUSEBUTTONDOWN:
            mouse_pos = pygame.mouse.get_pos()
            mouse_button = event.button
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

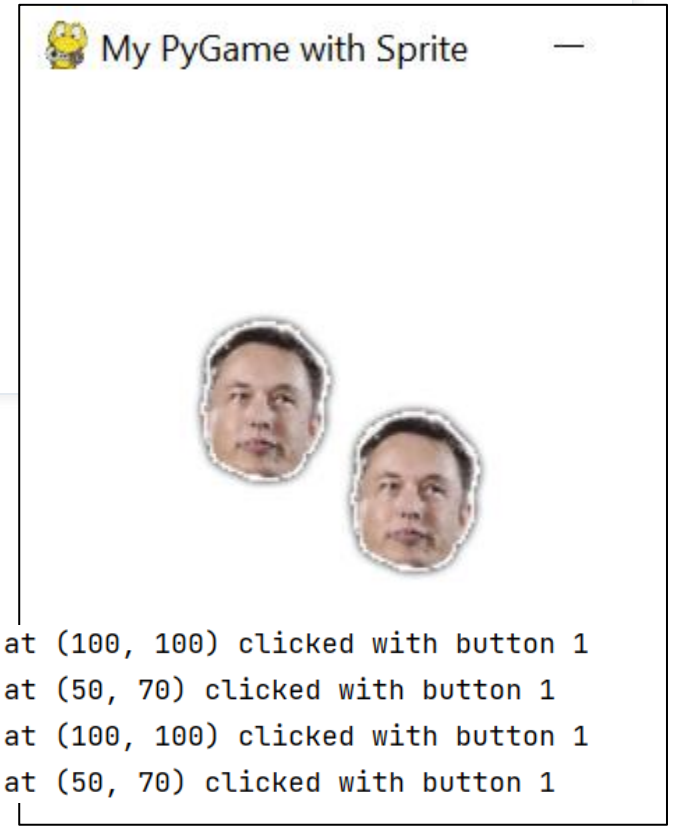
Ln 49, Col 34 Spaces: 4 UTF-8 CRLF Python 3.7.7 64-bit Prettier

Mouse events

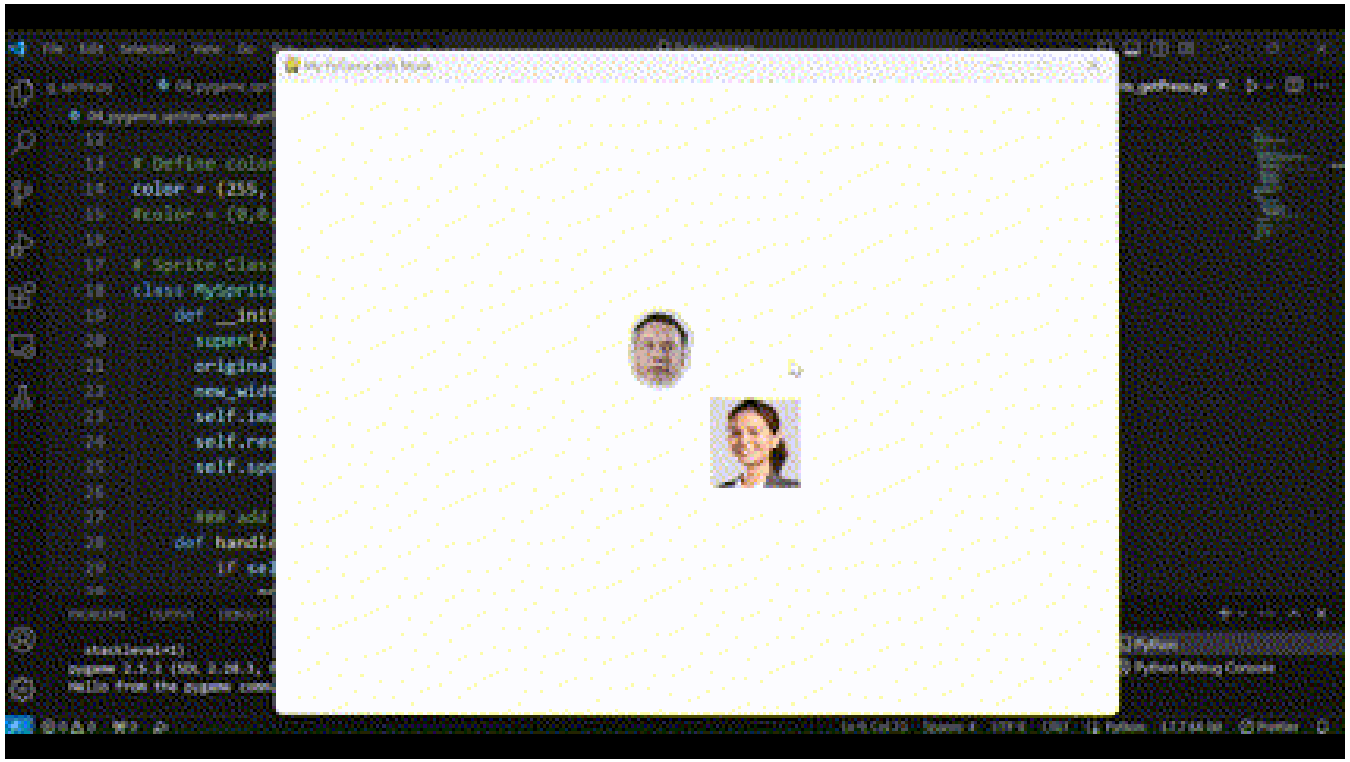
- Mouse actions lead to events with specific types:
  - press button down: `MOUSEBUTTONDOWN`
  - release button: `MOUSEBUTTONUP`
  - move the cursor: `MOUSEMOTION`

- At any point you can call `mouse.get_pos()` which returns the mouse's current position as an (x, y) tuple.

```
e = event.wait()
if e.type == MOUSEMOTION:
    pt = mouse.get_pos()
    x, y = pt
    ...
```



If the sprite is clicked, it will print a message.



Task#4: Separate movement for each sprite instance

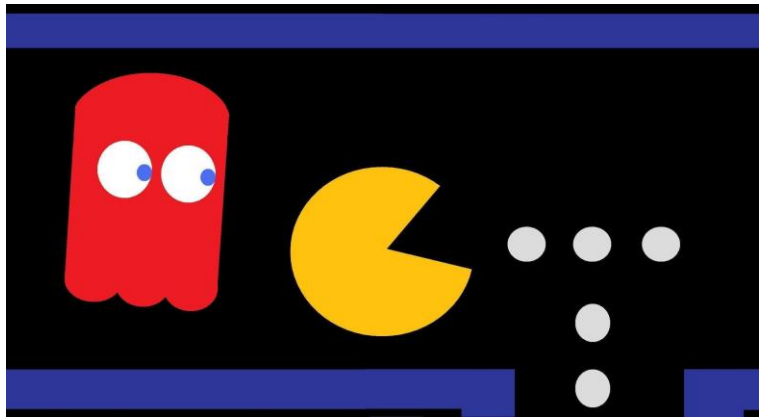
04_pygame_sprites_events_getPress_task04.py

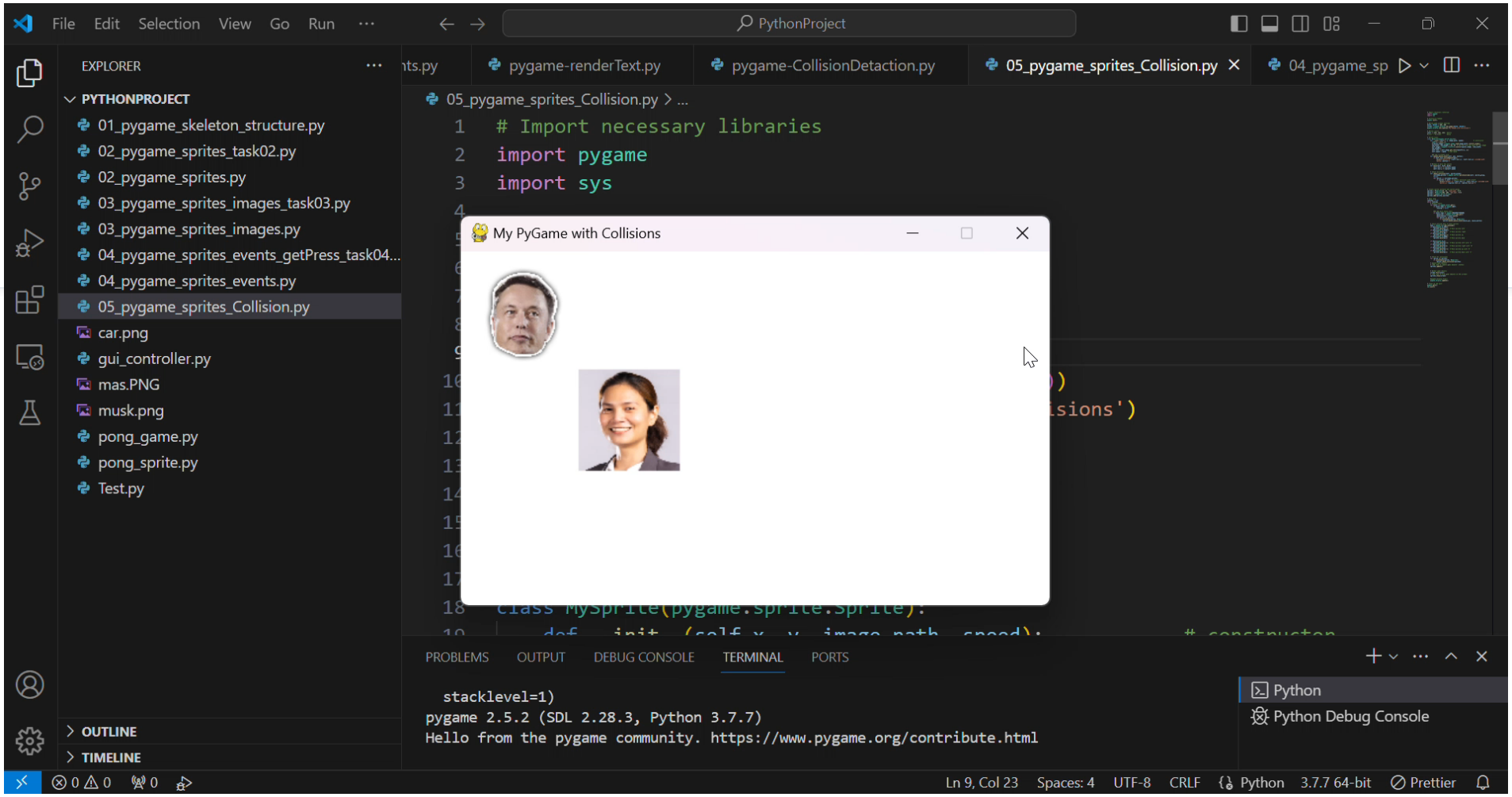
```
# Method to move the sprite
    def move(self, dx=0, dy=0):
        self.rect.x += dx*speed
        self.rect.y += dy*speed
```

```
# Check pressed keys to move sprites
   keys = pygame.key.get_pressed()
   if keys[pygame.K_LEFT]:
       sprite1.move(dx=-1)  # Move sprite1 left
   if keys[pygame.K_RIGHT]:
       sprite1.move(dx=1)   # Move sprite1 right
   if keys[pygame.K_UP]:
       sprite1.move(dy=-1)  # Move sprite1 up
   if keys[pygame.K_DOWN]:
       sprite1.move(dy=1)   # Move sprite1 down

   if keys[pygame.K_a]:
       sprite2.move(dx=-1)  # Move sprite2 left with 'A'
   if keys[pygame.K_d]:
       sprite2.move(dx=1)   # Move sprite2 right with 'D'
   if keys[pygame.K_w]:
       sprite2.move(dy=-1)  # Move sprite2 up with 'W'
   if keys[pygame.K_s]:
       sprite2.move(dy=1)   # Move sprite2 down with 'S'
```

Collision detection

- collision detection: Examining pairs of sprites to see if they are touching each other.
  - e.g. seeing whether sprites' bounding rectangles intersect
  - usually done after events occur,
    or at regular timed intervals
  - can be complicated and error-prone
    - optimizations: *pruning* (only comparing some sprites, not all), ...





[05_pygame_sprites_Collision.py]

Collisions btwn. rectangles

- Recall: Each `Sprite` contains a `Rect` collision rectangle stored as a field named `rect`
- `Rect` objects have useful methods for detecting collisions between the rectangle and another sprite:

`collidepoint(`**p**`)`	returns `True` if this `Rect` contains the point
`colliderect(`**rect**`)`	returns `True` if this `Rect` touches the rect

```
if sprite1.rect.colliderect(sprite2.rect):
    # they collide!
    ...
```

Collisions between groups

global pyGame functions to help with collisions:

`spritecollideany(`sprite, group`)`

- Returns `True` if sprite has collided with any sprite in the group

`spritecollide(`sprite, group, kill`)`

- Returns a list of all sprites in group that collide with sprite
- If kill is `True`, a collision causes sprite to be deleted/killed

`groupcollide(`group1, group2, kill1, kill2`)`

- Returns list of all sprites in group1 that collide with group2

Drawing text: Font

06_pygame_sprites_RenderText.py

- Text is drawn using a `Font` object:

  name `= Font(`filename`,` size`)`

  - Pass `None` for the file name to use a default font.

- A `Font` draws text as a `Surface` with its `render` method:

  name`.render("`text`", True, (`red`,` green`,` blue`))`

  Example:

```
my_font = Font(None, 16)
text = my_font.render("Hello", True, (0, 0, 0))
```

Displaying text

- A `Sprite` can be text by setting that text's `Surface` to be its `.image` property.

Example:

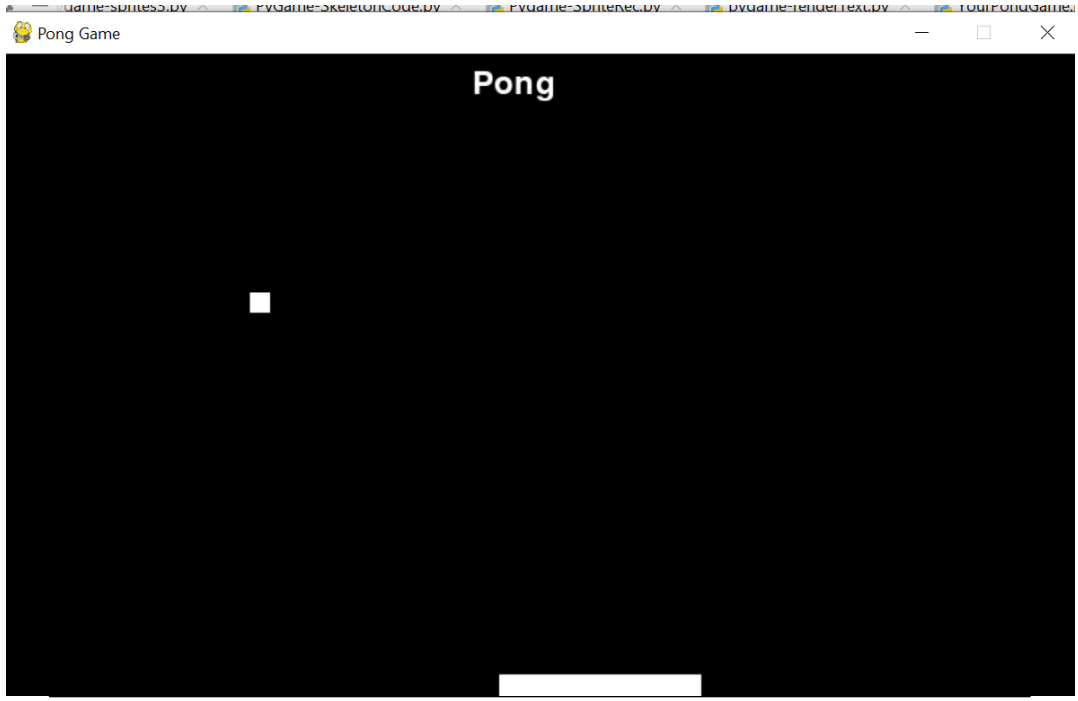
```
class Banner(Sprite):
    def __init__(self):
        my_font = Font(None, 24)
        self.image = my_font.render("Hello", True, \
                                    (0, 0, 0))
        self.rect = self.image.get_rect().move(50,70)
```

Task#5 Create a Ping Pong game

Exercise: Pong

- Let's create a Pong game with a bouncing ball and paddles.
  - 800x480 screen size
  - 15x15 square ball bounces off of any surface it touches
  - one 150x120 paddle move when holding Left/Right arrows
  - game displays score on top/center of screen in a 72px font

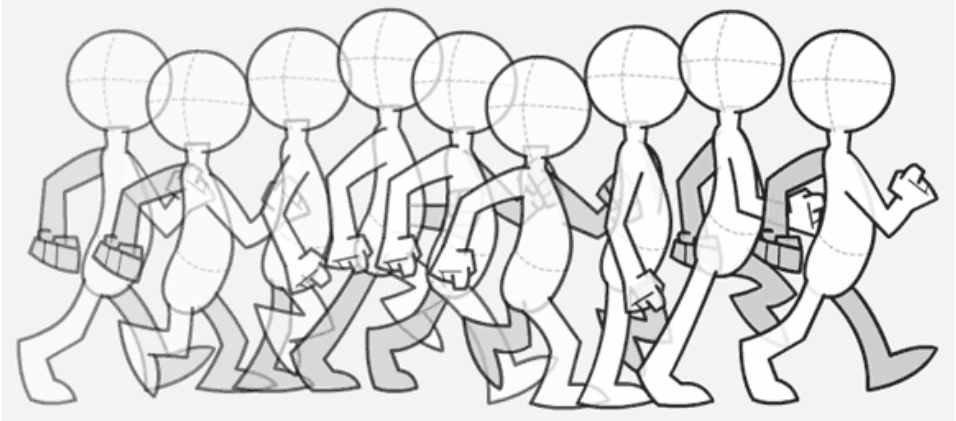
1. Create a new class for the ball, which will move and bounce off walls and paddles.
2. Modify the player paddle movement to only move vertically or horizontally.
3. Update the game loop to handle ball-paddle collisions and scoring.
4. Add scoring text to the screen.



07-pygame_ponggame.py

Animation

- Many action games, rather than waiting for key/mouse input, have a constant animation timer.
  - The timer generates events at regular intervals.
  - On each event, we can move/update all sprites, look for collisions, and redraw the screen.



Timer events

`time.set_timer(USEREVENT,` delayMS`)`

- Animation is done using timers
  - Events that automatically occur every *delayMS* milliseconds; they will have a type of `USEREVENT`
  - Your event loop can check for these events. Each one is a "frame" of animation

```
while True:
    e = event.wait()
    if e.type == USEREVENT:
        # the timer has ticked
        ...
```

Key presses

- key presses lead to `KEYDOWN` and `KEYUP` events
- `key.get_pressed()` returns an array of keys held down
  - the array indexes are constants like `K_UP` or `K_F1`
  - values in the array are booleans (`True` means pressed)

  - Constants for keys: `K_LEFT`, `K_RIGHT`, `K_UP`, `K_DOWN`, `K_a` - `K_z`, `K_0` - `K_9`, `K_F1` - `K_F12`, `K_SPACE`, `K_ESCAPE`, `K_LSHIFT`, `K_RSHIFT`, `K_LALT`, `K_RALT`, `K_LCTRL`, `K_RCTRL`, ...

```
keys_down = key.get_pressed()
if keys_down[K_LEFT]:
    # left arrow is being held down
    ...
```

Updating sprites

```
class name(Sprite):
    def __init__(self):
        ...

    def update(self):  # right by 3px per tick
        self.rect = self.rect.move(3, 0)
```

- Each sprite can have an `update` method that describes how to move that sprite on each timer tick.
  - Move a rectangle by calling its `move(dx, dy)` method.
  - Calling `update` on a `Group` updates all its sprites.

Sounds

- Loading and playing a sound file:

```
from pygame.mixer import *
mixer.init()          # initialize sound system
mixer.stop()          # silence all sounds

Sound("filename").play()  # play a sound
```

- Loading and playing a music file:

```
music.load("filename")     # load bg music file
music.play(loops=0)         # play/loop music
                         # (-1 loops == infinite)
```

others: `stop`, `pause`, `unpause`, `rewind`, `fadeout`, `queue`