Basic Python Programming

[Session 4] Pygame

Contents

- **Intro. to Pygame & Preparation**

Intro & Preparation

Pygame

- **A free Python library for video games**
  - Fully-optimized
  - Good portability and cross-platform

About Today's Class…

- **We will make several games for exercise**
  - Instead of many backgrounds

- **What you need to do in this class is learning about:**
  - How to use "a tool"
  - Overall mechanisms and procedures

- **So, just enjoy!**

Setting Environment

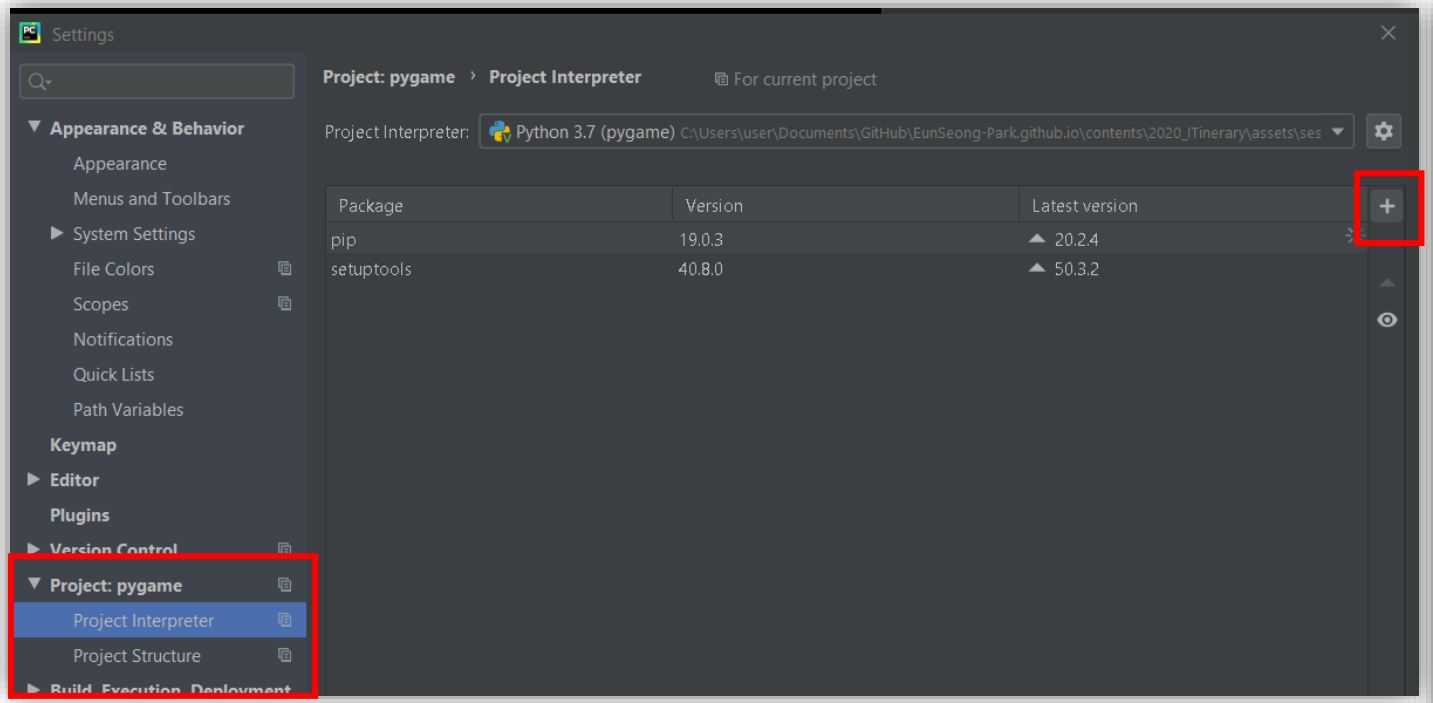
- **If you didn’t have any problems in the session 3, that’s ok.**

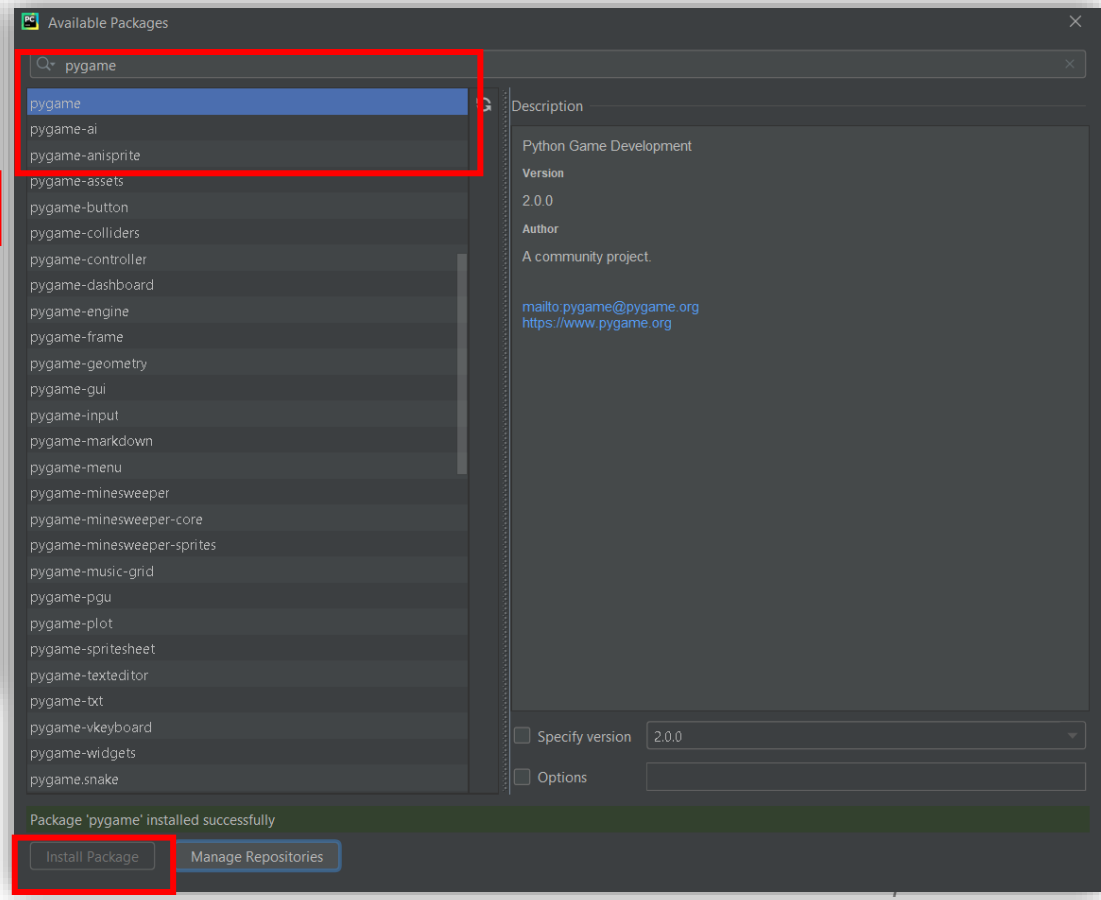
- **Pygame recommends to use Python 3.7.7 or greater**
  - We use 3.7.8

- **Supports various operating systems:**
  - Windows, Mac, Debian, Ubuntu, Mint, Raspberry Pi, etc.

Installing Pygame

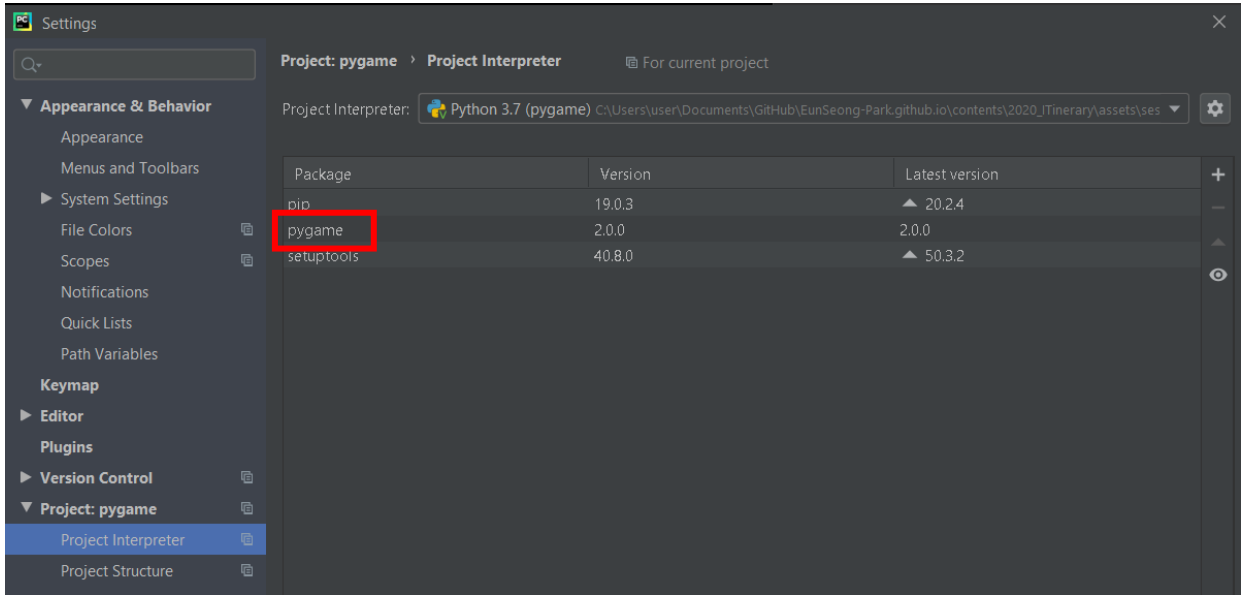
- Like installing OpenCV, just find “pygame” and install

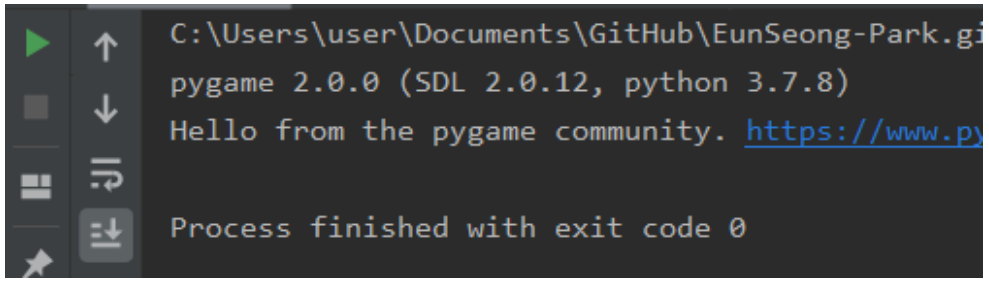




Check

- **Then you are ready for this class!**

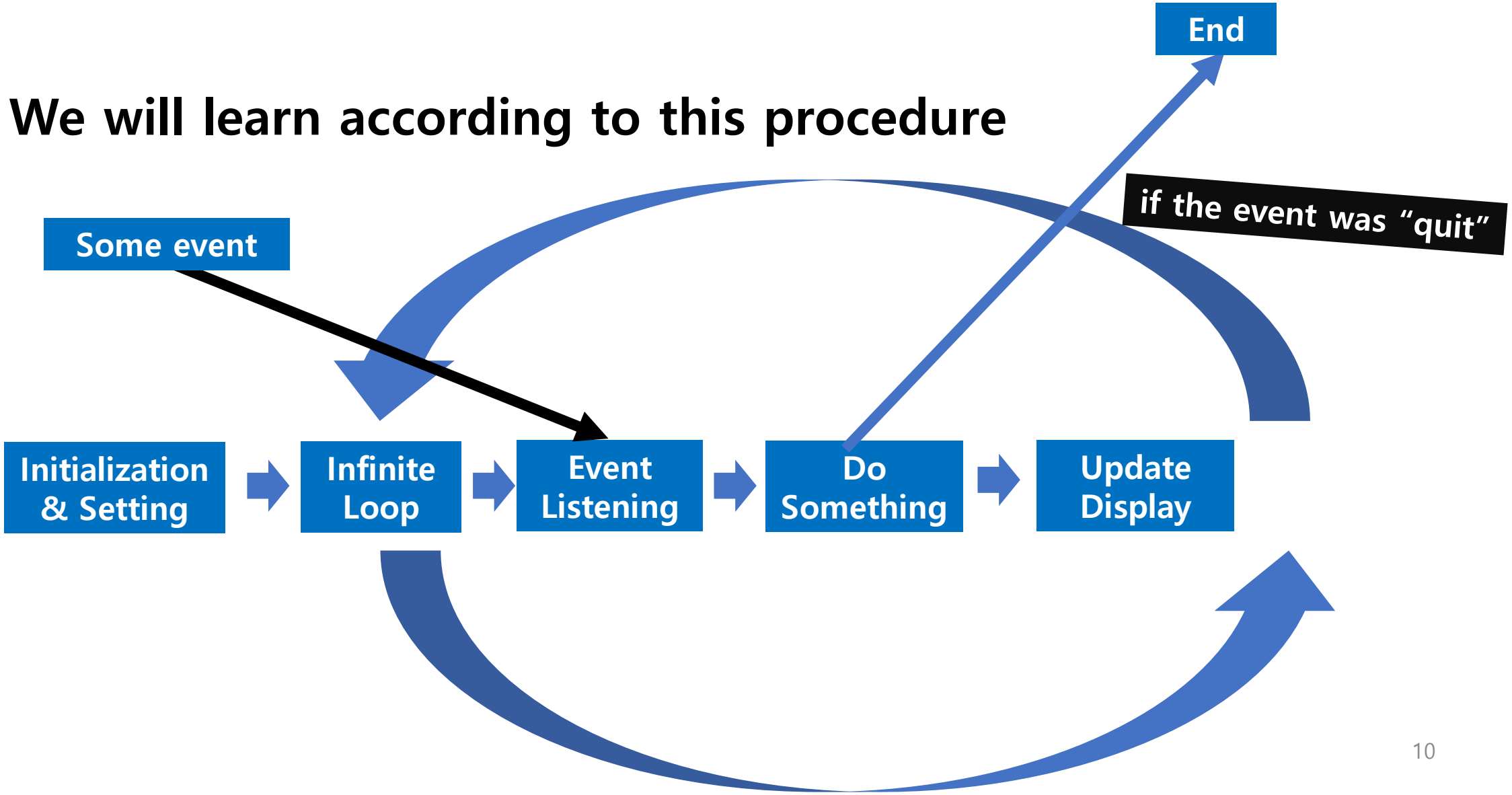




Basic Flow of Pygame

Basic Flow

- We will learn according to this procedure



Example: Chrome Dinosaur Game [1]

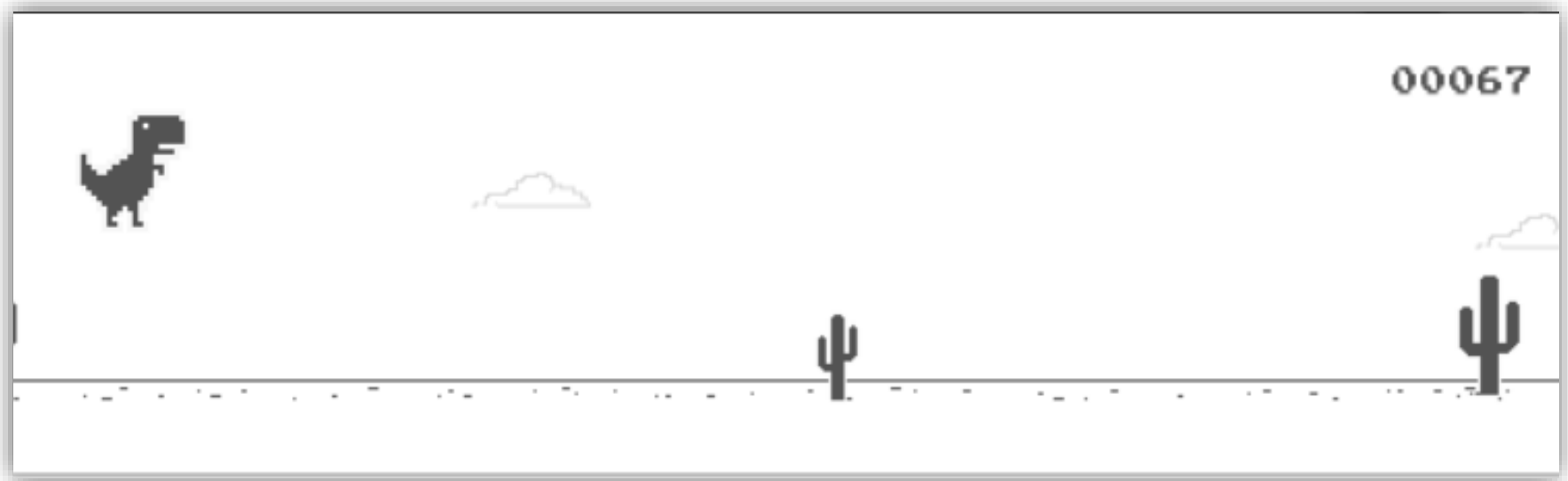
- **If you are using Google Chrome, try this after disconnecting your internet**
  - or… Use chrome extension without disconnecting
  - https://chrome.google.com/webstore/detail/running-dinosaur-game/nihmppmidbbbkfademfpjmhhogegjbjd/related

Example: Chrome Dinosaur Game [2]

- **Initialization**
  - Screen (window) setting
  - Setting dinosaur object
  - …

Example: Chrome Dinosaur Game [3]

- **Event listening / handling**
  - Spacebar is pressed? -> jump
  - Time is elapsed? -> increase score
  - collided with cactus? -> game over
  - …and regularly update screen



Example: Chrome Dinosaur Game [4]

- **Quit**
  - …If the connection is recovered…

- **Anyway, in many games, you can find these procedure**

Initialization: Module

- **Every module can be used after `pygame.init()`**
  - …and terminated by `pygame.quit()`

- **…is it over?**
  - no

Initialization: Surface(screen)

- **“Surface” is one of most important object**
  - It can be used to represent “image” (such as background)
  - You can imagine a “canvas”

- **`pygame.display.set_mode((w, h))` initializes a window(screen)**
  - And returns the corresponding Surface object
  - We implicitly set the screen size here

- **(Optional) You can set the caption of window (title)**
  - pygame.display.set_caption(“title”)

Initialization: Clock

- **Why we need a clock?**
  - to measure time elapsed
  - to implement periodical event
  - to update screen periodically
  - so on…

- **Creation is very simple:**

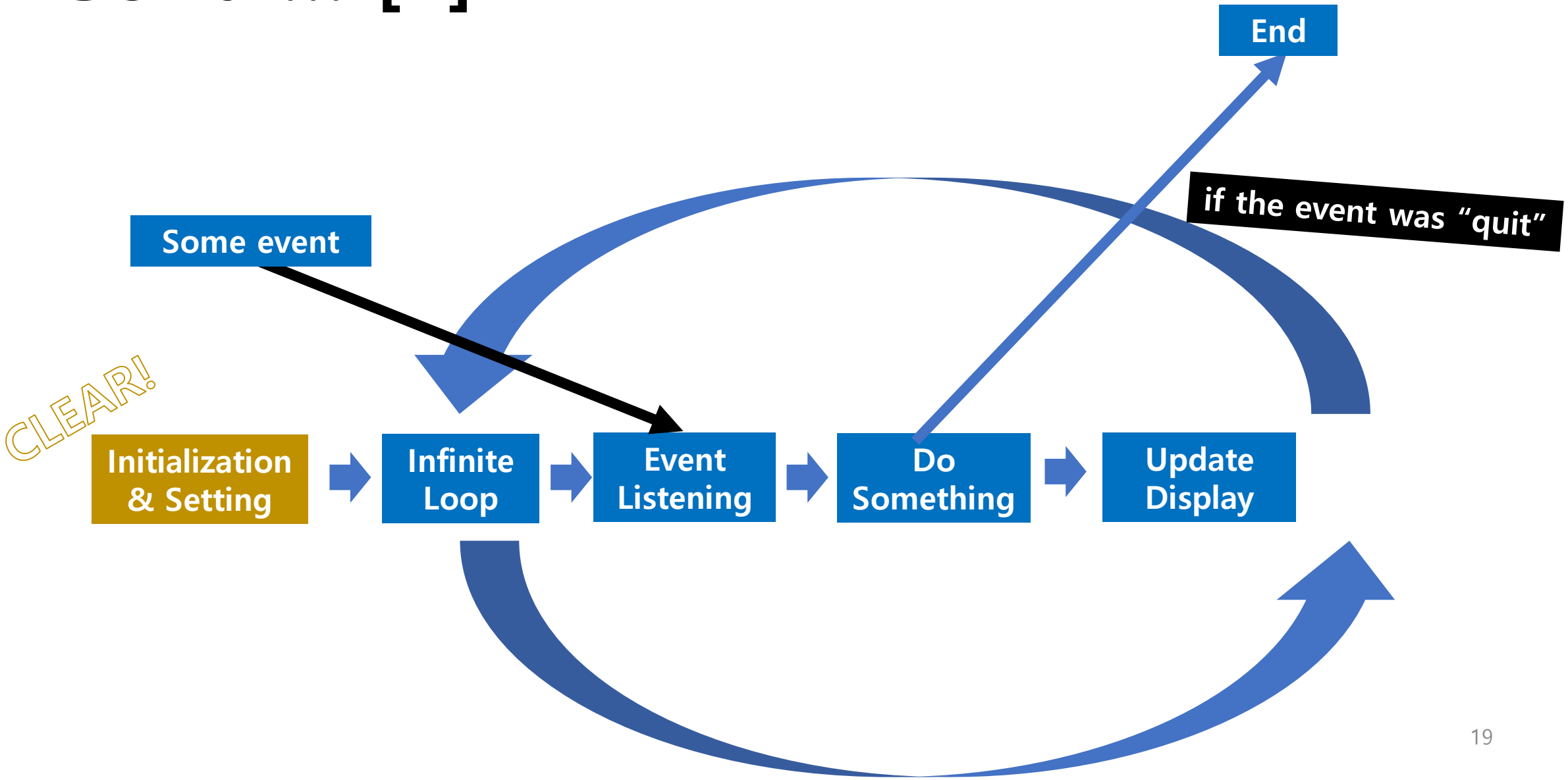
```
clock = pygame.time.Clock()
```

  - We will learn how to use it a little later

So far… [1]

```python
import pygame

pygame.init()

screen = pygame.display.set_mode((640, 480))
clock = pygame.time.Clock()
```

So far… [2]


End
if the event was "quit"
Some event
CLEAR!
Initialization & Setting
Infinite Loop
Event Listening
Do Something
Update Display

Loop [1]

- **First, implement showing empty screen with a fixed FPS**

- **`(Surface).fill("color")` fills surface with the color**
  - "color" is given by 3-tuple(R,G,B)

- **`pygame.display.flip()` updates and shows the screen to us**

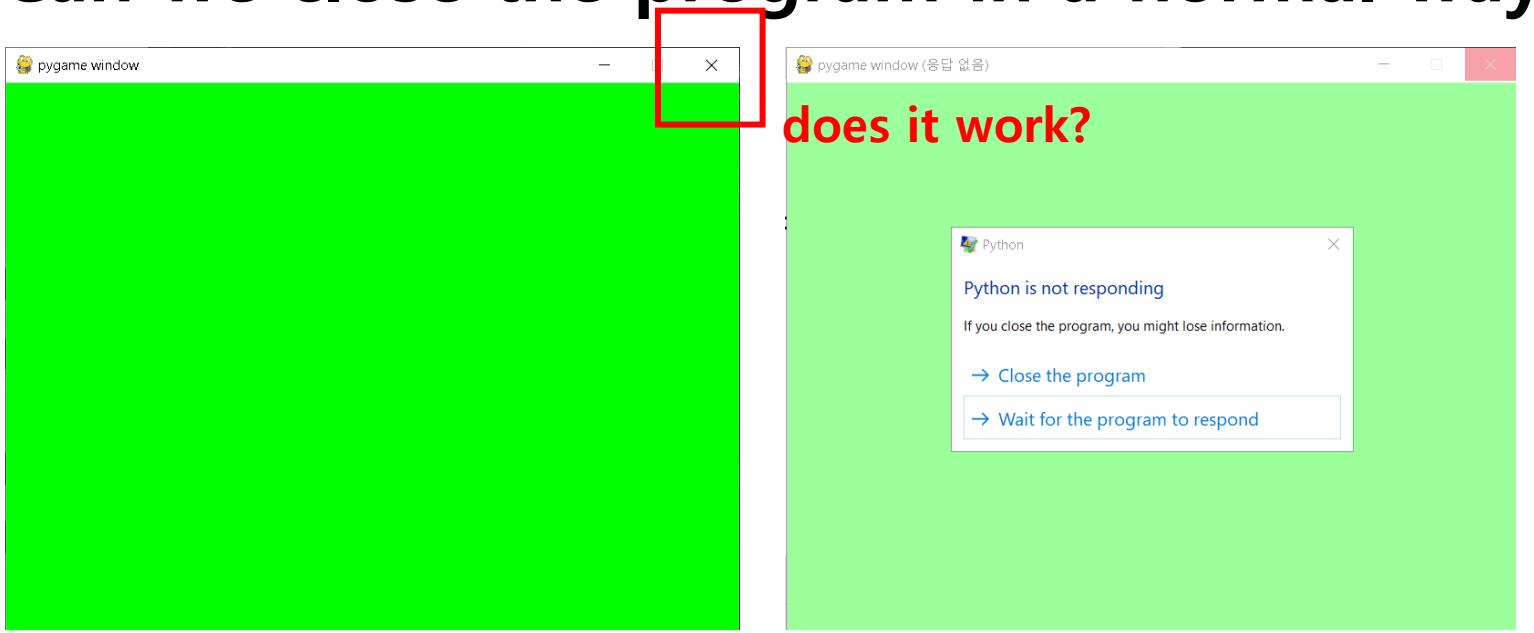
- **`(clock).tick("msec")` waits for "msec"**
  - Why do we need to wait?

Loop [2]

- Let's run… but somewhat goes wrong

```python
while True:
    screen.fill((255, 0, 0))
    pygame.display.flip()
    clock.tick(30)
```

- Can we close the program in a normal way?



Event Handling [1]

- **Pygame "listens" events such as:**
  - Mouse move, click, …
  - Keyboard input
  - Window activation
  - "Close window" button
  - so on…

- **We can access to the occurred events by `pygame.event.get()`**
  - We use `for` statement… why?

Event Handling [2]

- **Basic use**

```
8   while True:
9       for event in pygame.event.get():
10          if event.type == "Something":
11              "do something"
12          elif event.type == "Something else":
13              "do something else"
14          "..."
15
16      screen.fill((0, 255, 0))
17      pygame.display.flip()
18      clock.tick(30)
```

Event Handling: QUIT

- **First, we should handle the event: quit**
  - "Close button"
  - Ctrl+C in Linux
- **To exit the program, import sys module:**
  - `sys.exit()` terminates the program

```
import pygame
import sys
```

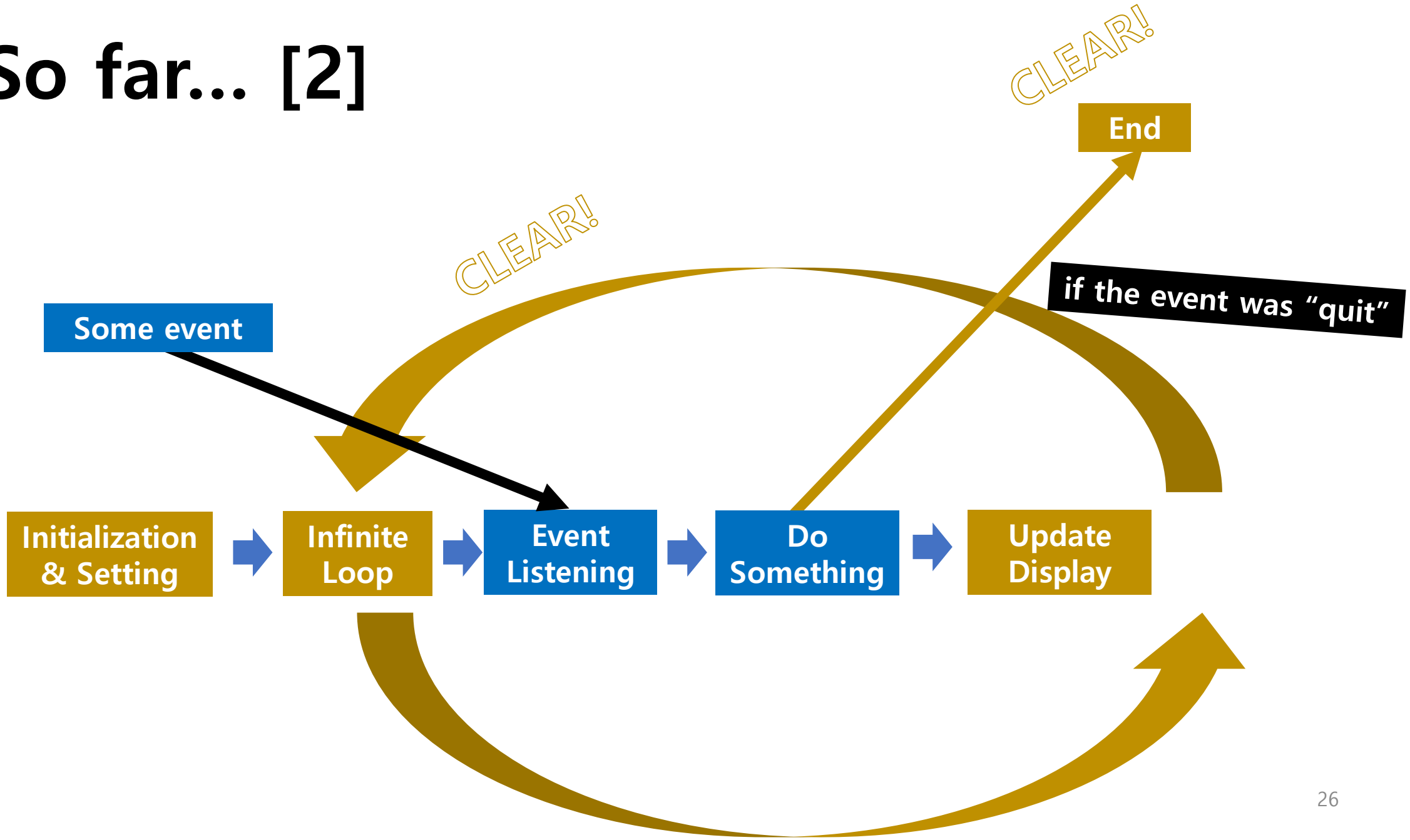
- **The event type is `pygame.QUIT`**
  - So…

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    screen.fill((0, 255, 0))
    pygame.display.flip()
    clock.tick(30)
```

So far… [1]

```python
import pygame
import sys

pygame.init()

screen = pygame.display.set_mode((640, 480))
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    screen.fill((0, 255, 0))
    pygame.display.flip()
    clock.tick(30)
```

So far… [2]


CLEAR!
CLEAR!
End
if the event was "quit"
Some event
Initialization & Setting
Infinite Loop
Event Listening
Do Something
Update Display

Notes

- **What kinds of event, and how handle these?**
  - It entirely depends on our purpose
- **We will look at several commonly used methods, by examples**
- **So, of course, you may find a better way!**

Implementation Examples

Keyboard Input [1]

- **Many game uses keyboard input**
  - WASD / ←→↑↓ to move character
  - Esc for menu / to exit game
  - QWER to use skill (LoL)
  - so on…

- **Key input (key-up and key-down) is defined as an event**
  - we can access to the event by `pygame.event.get()` (like QUIT)
  - These are defined as `pygame.KEYUP` / `pygame.KEYDOWN`

Keyboard Input [2]

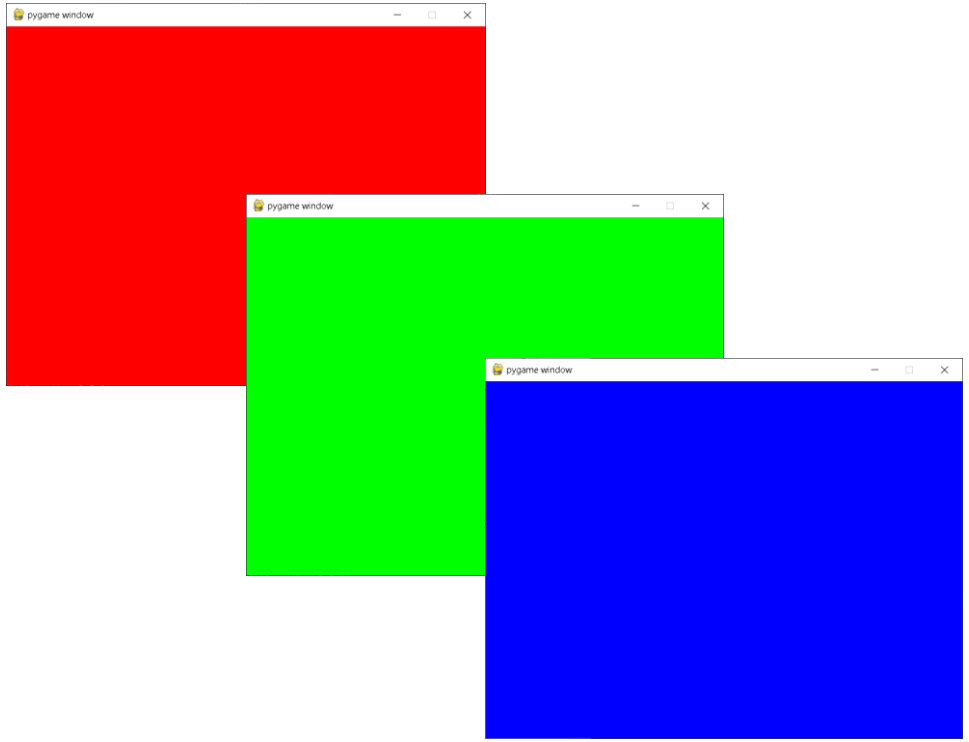
- **Each key corresponding with event is defined as `pygame.K_(key)`**
  - and can be accessed by `event.key`
  - For example, a -> pygame.K_a

- **So, the procedure is like…**
  - Listen event, is it keyboard input(up/down)?
  - Then, the corresponding key is what I want to use?
  - Then, do something!

Keyboard Input [3]

- Example: press R to screen to red, G to green, B to blue

```python
import pygame
import sys

pygame.init()
screen = pygame.display.set_mode((640, 480))
clock = pygame.time.Clock()
color = (0, 0, 0)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_r:
                color = (255, 0, 0)
            elif event.key == pygame.K_g:
                color = (0, 255, 0)
            elif event.key == pygame.K_b:
                color = (0, 0, 255)

    screen.fill(color)
    pygame.display.flip()
    clock.tick(30)
```



Drawing Shapes [1]

- **We need to draw something in the screen:**
  - Character?
  - User interface?
  - Message?
  - …

- **Pygame provides various shapes to draw**
  - and these are similar with those in OpenCV

Drawing Shapes [2]

- **We use `pygame.draw.(shape)()` function**
  - Let's see the examples (these are very intuitive!)
  - Many other shapes are explained in the supplement

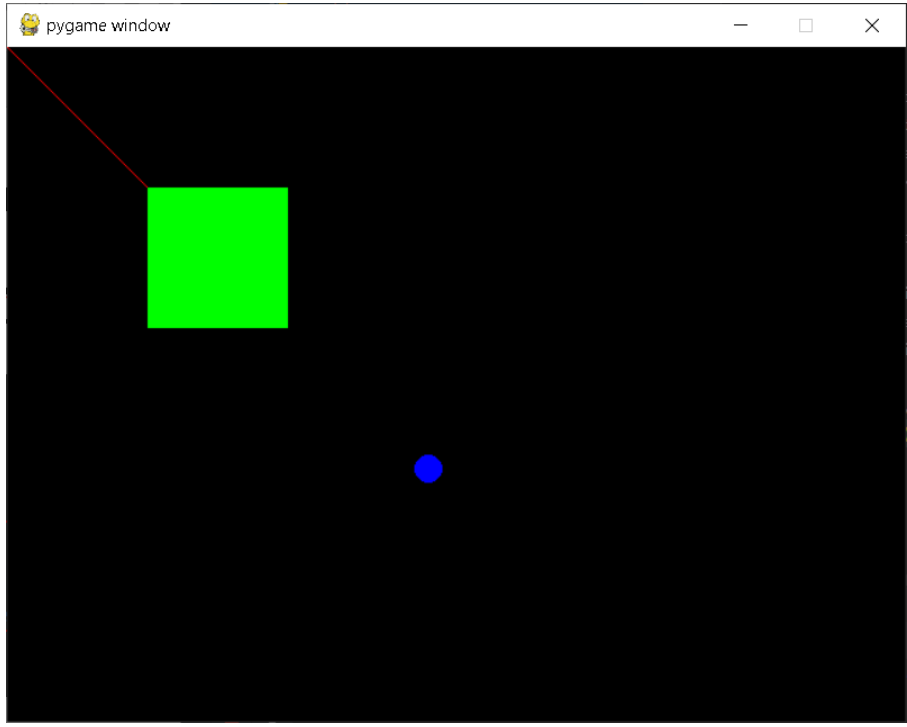
```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()

    screen.fill(color)

    pygame.draw.line(screen, (255, 0, 0), (0, 0), (100, 100), width=1)
    pygame.draw.rect(screen, (0, 255, 0), (100, 100, 100, 100), width=0)
    pygame.draw.circle(screen, (0, 0, 255), (300, 300), 10, width=0)

    pygame.display.flip()
    clock.tick(30)
```

shapes must be drawn after background is drawn

surface color start end

surface color (x, y, w, h)

surface color center radius



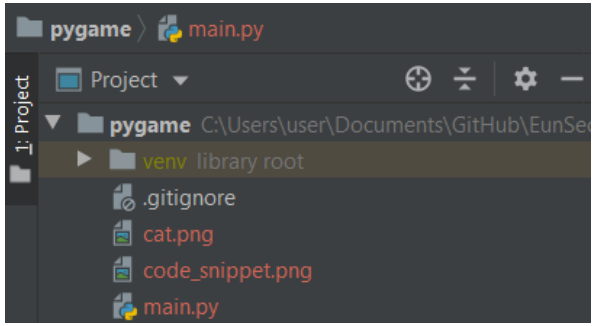
`blit()`: Showing image [1]

- **A surface's method, `blit()` copies given visual object into a certain location**
  - `blit("something", "location")`
  - "Location" can be 2-tuple(x, y), 4-tuple(x, y, w, h), or Rect (we will cover soon)
  - "Something" can be image, rendered text, or something else

blit(): Showing image [2]

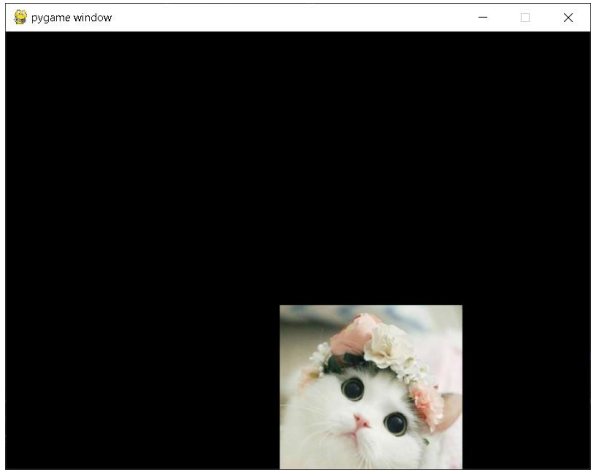
- First, load image by using pygame.image.load("filename")

```
13    img = pygame.image.load("cat.png")
```



- And then, use (Surface).blit("image", "Rect")

```
49        screen.blit(img, (200, 200, 300, 300))
```



`blit()`: Rendered Text [1]

- **We may want to show some text or number:**
  - Score / Point
  - Name
  - Description
  - …

- **Then, how?**

blit(): Rendered Text [2]

- **First, specify the "font":**
  - pygame.font.Font("filename", "size")
    - If None, the pygame default font is loaded
  - pygame.font.SysFont("name", "size")
    - Bring from system font

```
my_font_1 = pygame.font.Font(None, 24)
my_font_2 = pygame.font.SysFont("Arial", 24)
```

  - These function returns font object

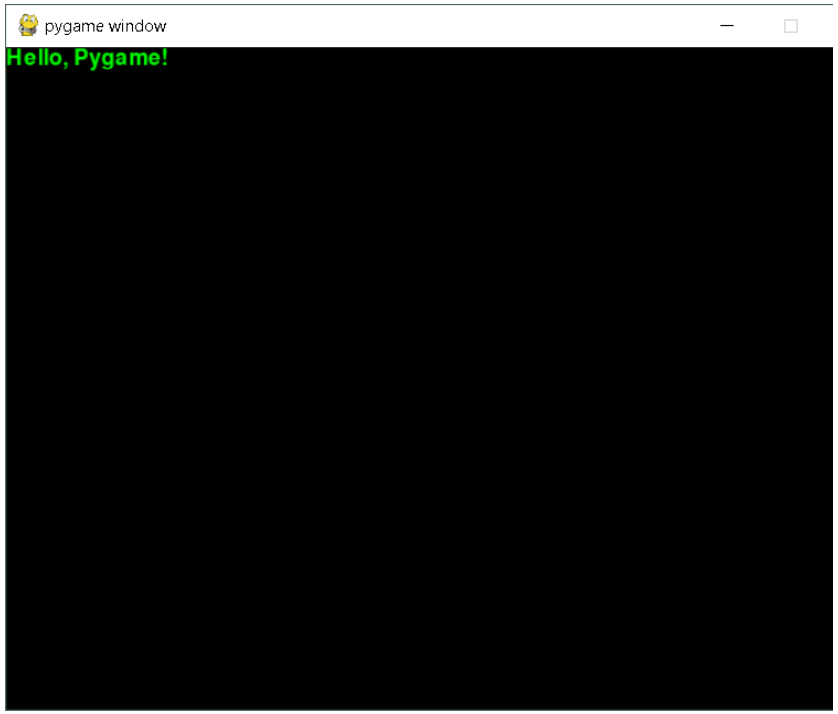
- **Note that the font size cannot be changed**

blit(): Rendered Text [3]

- **We use `render()` method to draw text**
  - `(Font).render(text, antialias, color)`

- **How can we use it?**
  - "blit" this result at a certain region

```
49      screen.fill(color)
50      text1 = my_font_1.render("Hello, Pygame!", True, (0,255,0))
51      screen.blit(text1, (0, 0, 200, 100))
52      pygame.display.flip()
53      clock.tick(60)
```



Note

- **In default, the object is not fitted to the region automatically.**
  - Neither extended nor compressed
  - Obviously, it is not 1x1 pixel

```
text1 = my_font_1.render("Hello, Pygame!", True, (0,255,0))
screen.blit(text1, (0, 0, 1, 1))
```

Mouse Event Handling

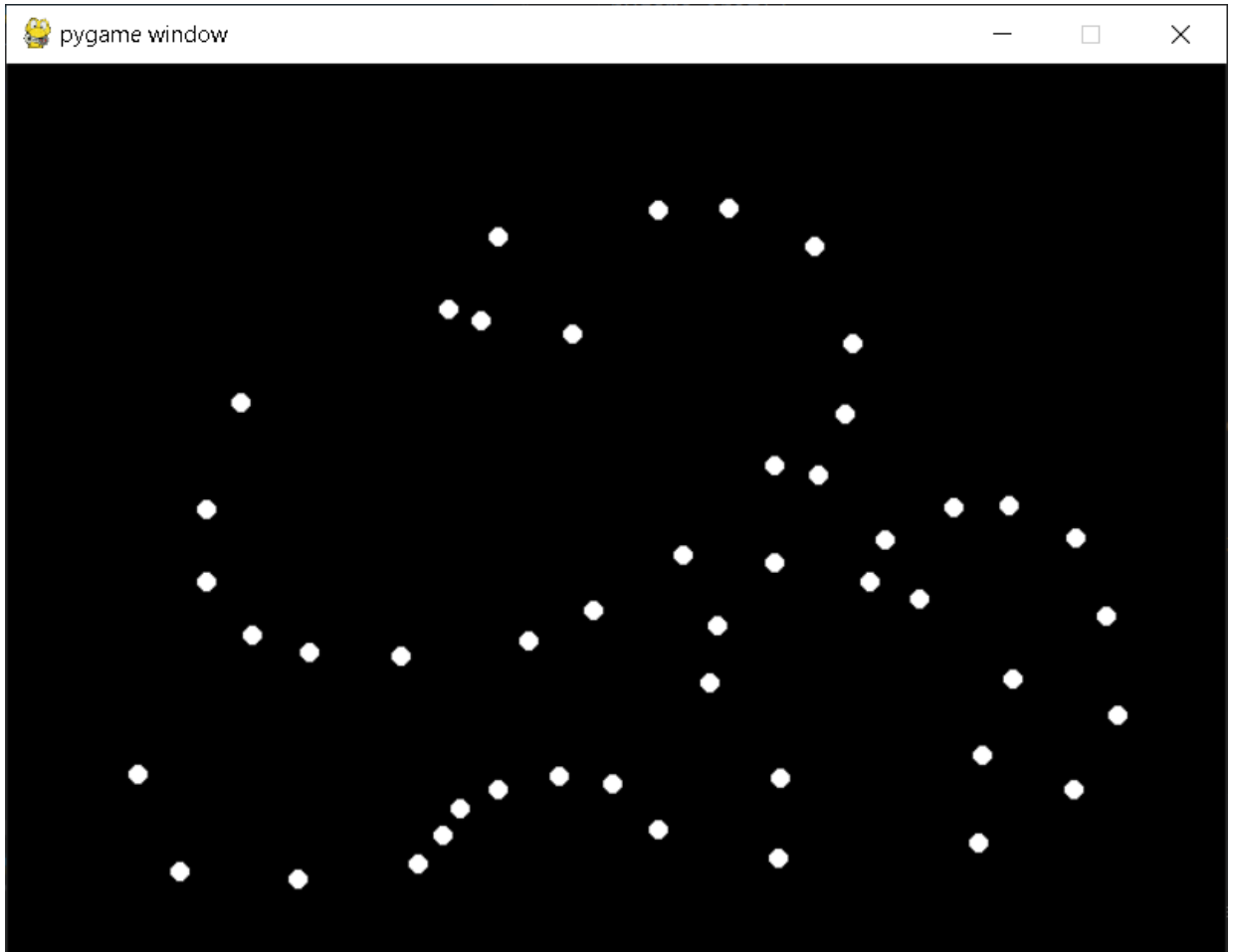
- **There are several events about mouse motion**
  - button-down (`pygame.MOUSEBUTTONDOWN`)
  - button-up (`pygame.MOUSEBUTTONUP`)
  - motion (`pygame.MOUSEMOTION`)

- **You can get the current mouse position with `pygame.mouse.get_pos()`**
  - It returns 2-tuple (x, y)

Mouse Event Handling: Example [1]

- Easiest, but naïve way

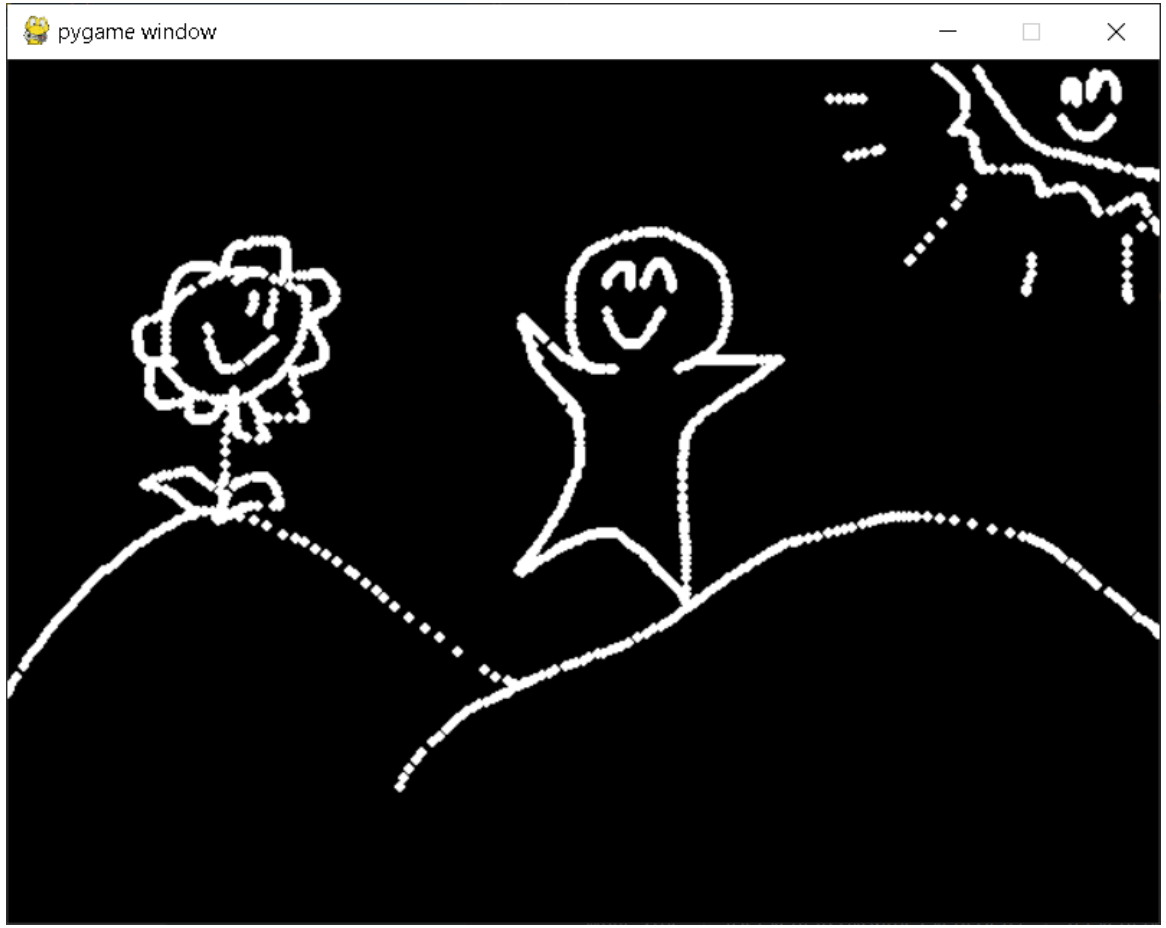
```python
import pygame
import sys

pygame.init()
screen = pygame.display.set_mode((640, 480))
clock = pygame.time.Clock()
color = (0, 0, 0)

screen.fill(color)
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.circle(screen,
                               (255, 255, 255),
                               pygame.mouse.get_pos(),
                               5,
                               width=0)

    pygame.display.flip()
    clock.tick(30)
```



Mouse Event Handling: Example [2]

- **Better way**

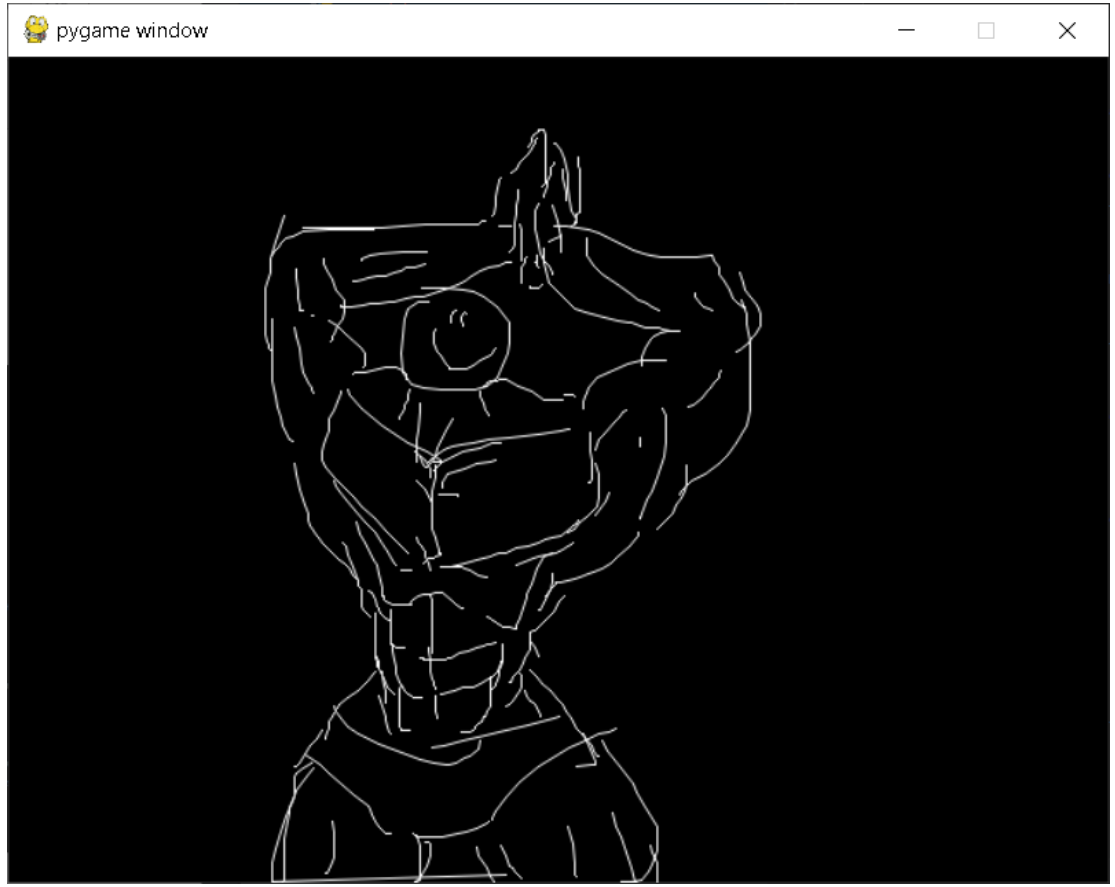
```python
color = (0, 0, 0)

mouse_toggle = False
screen.fill(color)
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if not mouse_toggle:
                mouse_toggle = True
        if event.type == pygame.MOUSEBUTTONUP:
            if mouse_toggle:
                mouse_toggle = False
        if event.type == pygame.MOUSEMOTION and mouse_toggle:
            pygame.draw.circle(screen,
                               (255, 255, 255),
                               pygame.mouse.get_pos(),
                               3,
                               width=0)
    pygame.display.flip()
    clock.tick(60)
```



Mouse Event Handling: Example [2]

- Much better way… but I am not good at drawing :(

```python
mouse_toggle = False
last_mouse_pos = (0, 0)

screen.fill(color)
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.MOUSEBUTTONDOWN:
            if not mouse_toggle:
                mouse_toggle = True
                last_mouse_pos = pygame.mouse.get_pos()
        if event.type == pygame.MOUSEBUTTONUP:
            if mouse_toggle:
                mouse_toggle = False
        if event.type == pygame.MOUSEMOTION and mouse_toggle:
            pygame.draw.aaline(screen,
                               (255, 255, 255),
                               last_mouse_pos,
                               pygame.mouse.get_pos(), True)
            last_mouse_pos = pygame.mouse.get_pos()

    pygame.display.flip()
    clock.tick(60)
```



In the Lab Session…

- **We will make two applications:**
  - Paint tool
  - Falling poop game

- **The previous example and the supplement may be helpful**

Rect Object

- **One of most important object in Pygame, as well as Surface**

- **It does..**
  - Indicate a certain region
    - in many cases, 4-tuple parameter can be replaced with Rect
  - Interact with another Rect
  - Represent an image
  - …

- **It can be regarded as a “sprite”**

Creating Rect

- **Use `pygame.Rect(x, y, w, h)`**

```
my_rect = pygame.Rect(200, 200, 50, 50)
```

- **…but it doesn’t do anything**
  - If you do not use it
  - So how can we use?

Moving Rect [1]

- **Two ways:**
  - Directly change each x, y of Rect
    - `(SomeRect).x = 10,`
    - `(SomeRect).y += 10`
    - …
  - Use `move(x, y)` method
    - (SomeRect).move(Δx, Δy)

Moving Rect [2]

- **Simple way:**
  - Cannot move continuously

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                my_rect = my_rect.move(-10, 0)
            if event.key == pygame.K_RIGHT:
                my_rect = my_rect.move(10, 0)
            if event.key == pygame.K_UP:
                my_rect = my_rect.move(0, -10)
            if event.key == pygame.K_DOWN:
                my_rect = my_rect.move(0, 10)

    screen.fill(color)
    screen.blit(img, my_rect)
    pygame.display.flip()
    clock.tick(60)
```

Moving Rect [3]

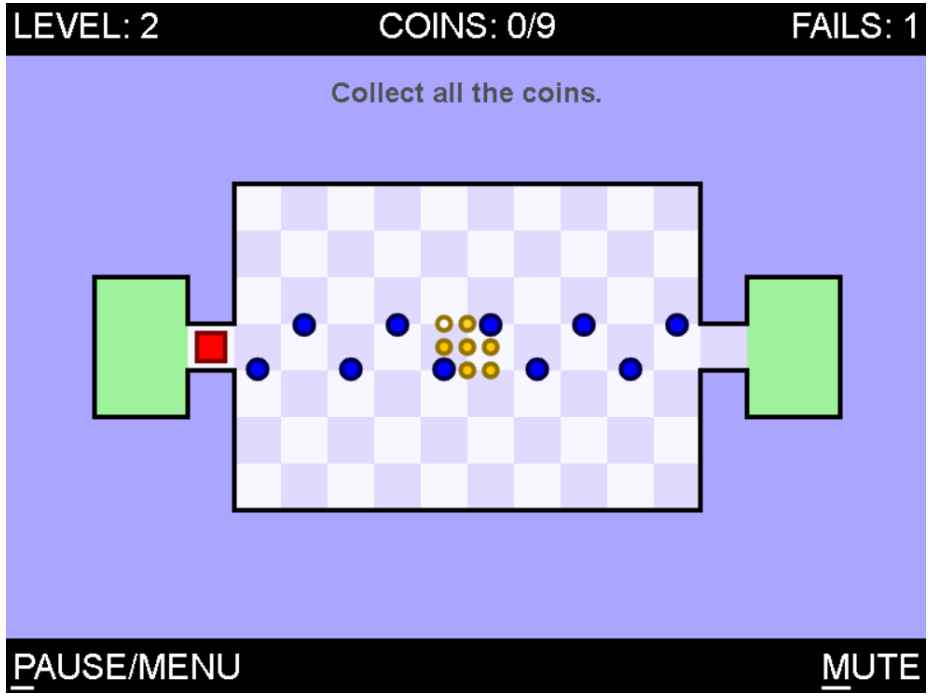
- **Better way:**
  - Can move continuously
  - Diagonal move (using 2 keys)

```python
to_move = [0, 0]
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                to_move[0] += -10
            if event.key == pygame.K_RIGHT:
                to_move[0] += 10
            if event.key == pygame.K_UP:
                to_move[1] += -10
            if event.key == pygame.K_DOWN:
                to_move[1] += 10

        if event.type == pygame.KEYUP:
            if event.key == pygame.K_LEFT:
                to_move[0] -= -10
            if event.key == pygame.K_RIGHT:
                to_move[0] -= 10
            if event.key == pygame.K_UP:
                to_move[1] -= -10
            if event.key == pygame.K_DOWN:
                to_move[1] -= 10

    my_rect = my_rect.move(to_move[0], to_move[1])
```

Collision [1]

- **In many games, two objects collide with each other**
  - Enemy's attack hits my body
  - My character is blocked by wall (so cannot move)
  - Some ball bounces to the ground
  - …

You know?



Collision [2]

- **In pygame, use `(Rect).collide_X("Something")` method**
  - `(Rect).collidepoint(x,y)`: check collision between the rect and point
  - `(Rect).colliderect(rect)`: check collision between the rects
  - `(Rect).collidelist(list)`: check collision between the rect and each elements in the list
- **We can use these in many ways**

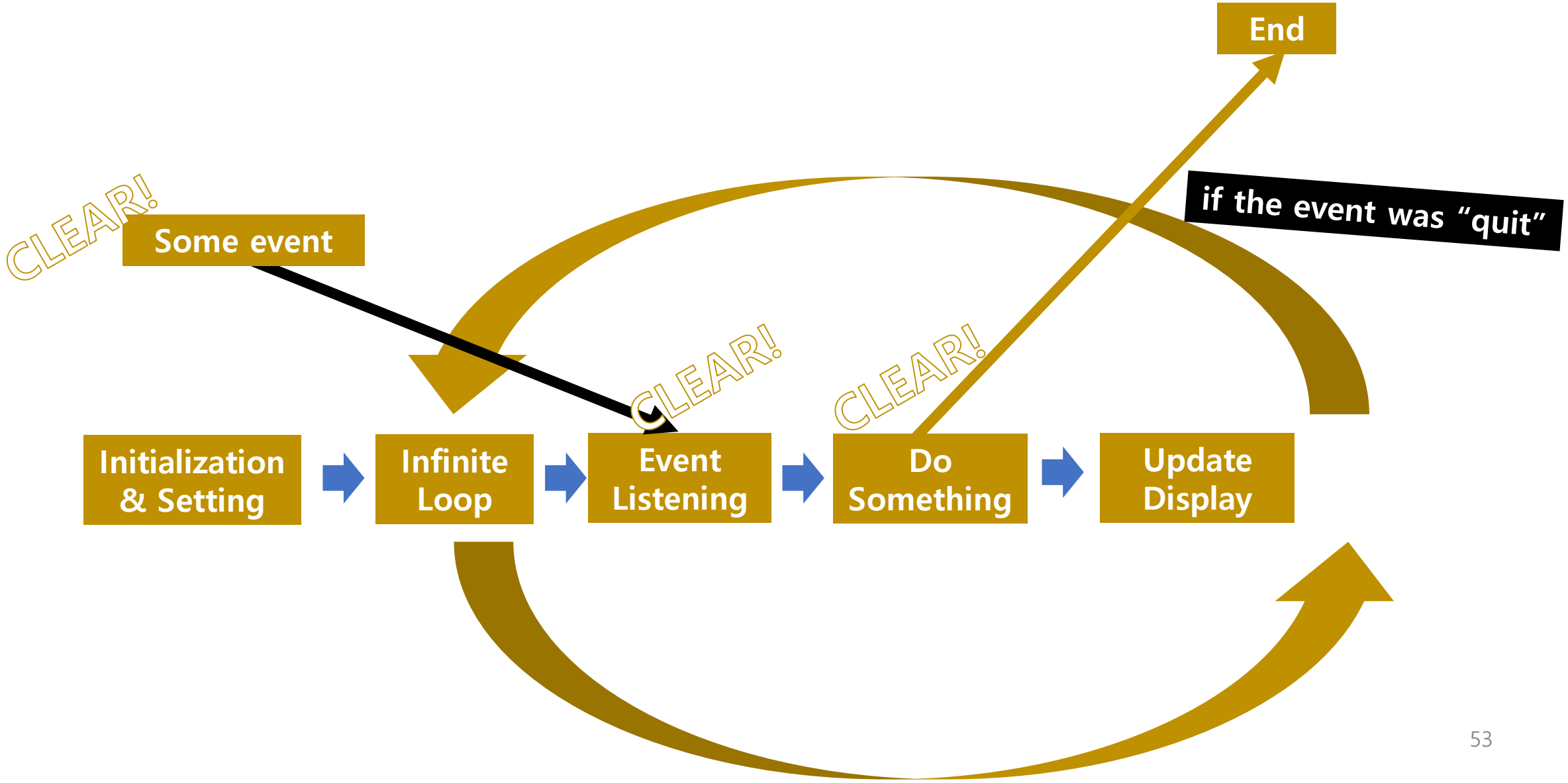
```
44    if my_rect.colliderect(my_enemy): # GAME OVER
45        print("meow")
46        pygame.quit()
47        sys.exit()
```

RE: In the Lab Session…

- **We will make two applications:**
  - Paint tool
  - Falling poop game

- **The previous example and the supplement may be helpful**

So far…


End
if the event was "quit"
CLEAR!
Some event
CLEAR!
CLEAR!
Initialization & Setting
Infinite Loop
Event Listening
Do Something
Update Display

About Pygame

- **Several useful functions and concepts are explained in the supplement material**

- **You can find an official documentation in here:**
  - https://www.pygame.org/docs/

Thank you