discover the basics of the Pygame library

What’s the goal of our meeting?

Quick intro

Python programming is a notably very good choice for developers for immediate prototyping of video games.

There are many libraries available in Python that help us to create cross-platform apps and games that can be used on both Android and iOS devices.

Top 5 Python game Engines:

- Pygame
- PyKyra
- Pyglet
- PyOpenGL
- Kivy

Short overview of game engines

All these engines share the following criteria:

- They’re relatively popular engines, or they cover aspects of gaming that aren’t usually covered.
- They’re currently maintained.
- They have good documentation available.

Today we are talking about Pygame.

What is the Pygame?

It is one of the oldest and most famous libraries of Python for game development. It is a set of Python modules designed for developing amazing games.

Features of Pygame:

- Cross-platform library
- Suitable for creating client-side applications
- Can use multicore CPUs
- Generate big results in a small amount of code

Okay, let’s discover Pygame

Create a screen

Pygame has a single display Surface that is either contained in a window or runs full screen.

Once you create the display you treat it as a regular Surface.

How to create a display surface?

```
# Create a display surface and set its caption
WINDOW_WIDTH = 600
WINDOW_HEIGTH = 600
display_surface = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGTH))
pygame.display.set_caption("Drawing Objects")
```

The core of all Pygame projects:

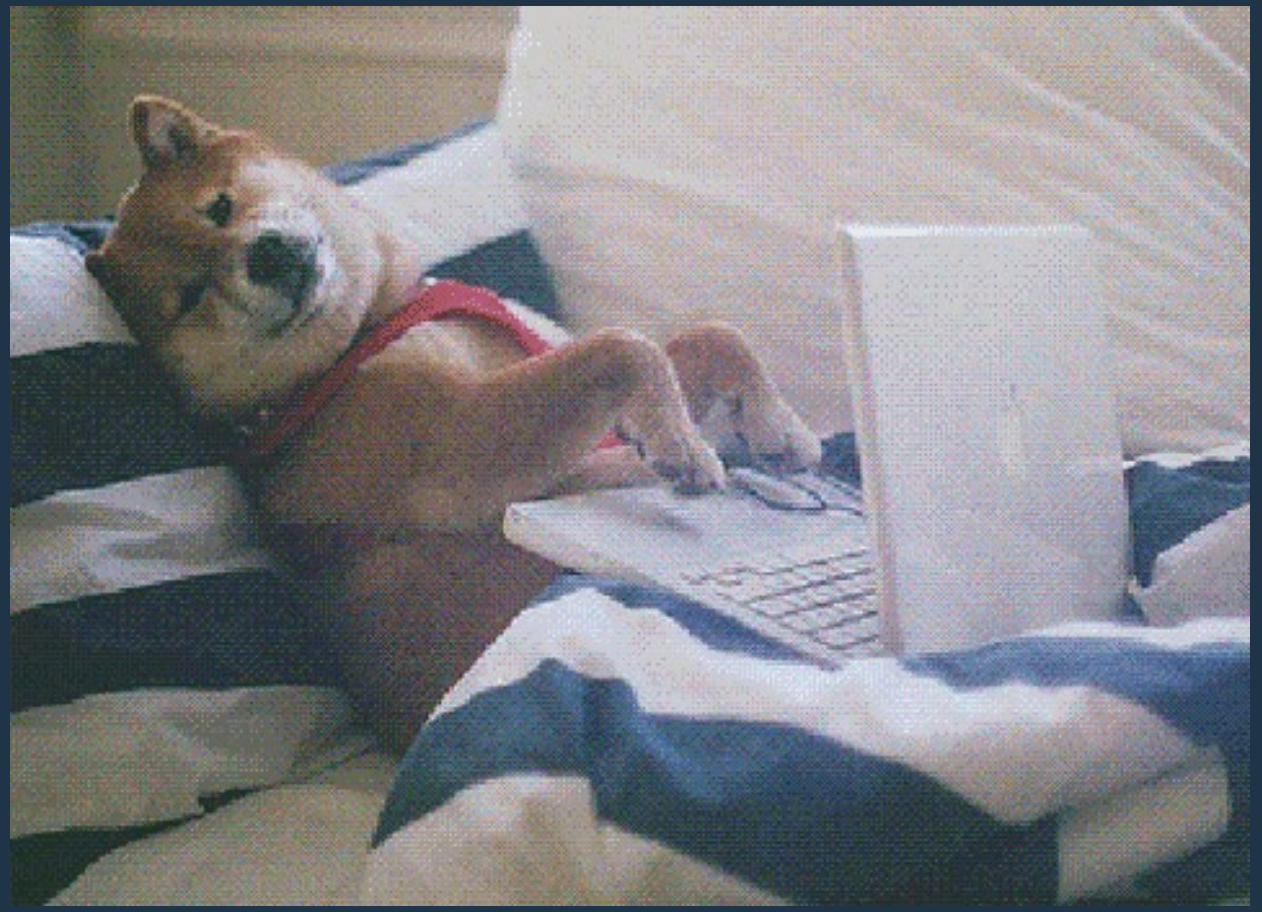
```python
import pygame

#INITIALIZE pygame
pygame.init()
WINDOW_WIDTH = 600
WINDOW_HEIGHT = 300
display_surface = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
pygame.display.set_caption("Hello Pygame!")

#The main game loop
running = True
while running:
    #Loop through a list of Event objects that have occured
    for event in pygame.event.get():
        print(event)
        if event.type == pygame.QUIT:
            running = False

    #End the game
pygame.quit()
```

Drawing figures on a display surface

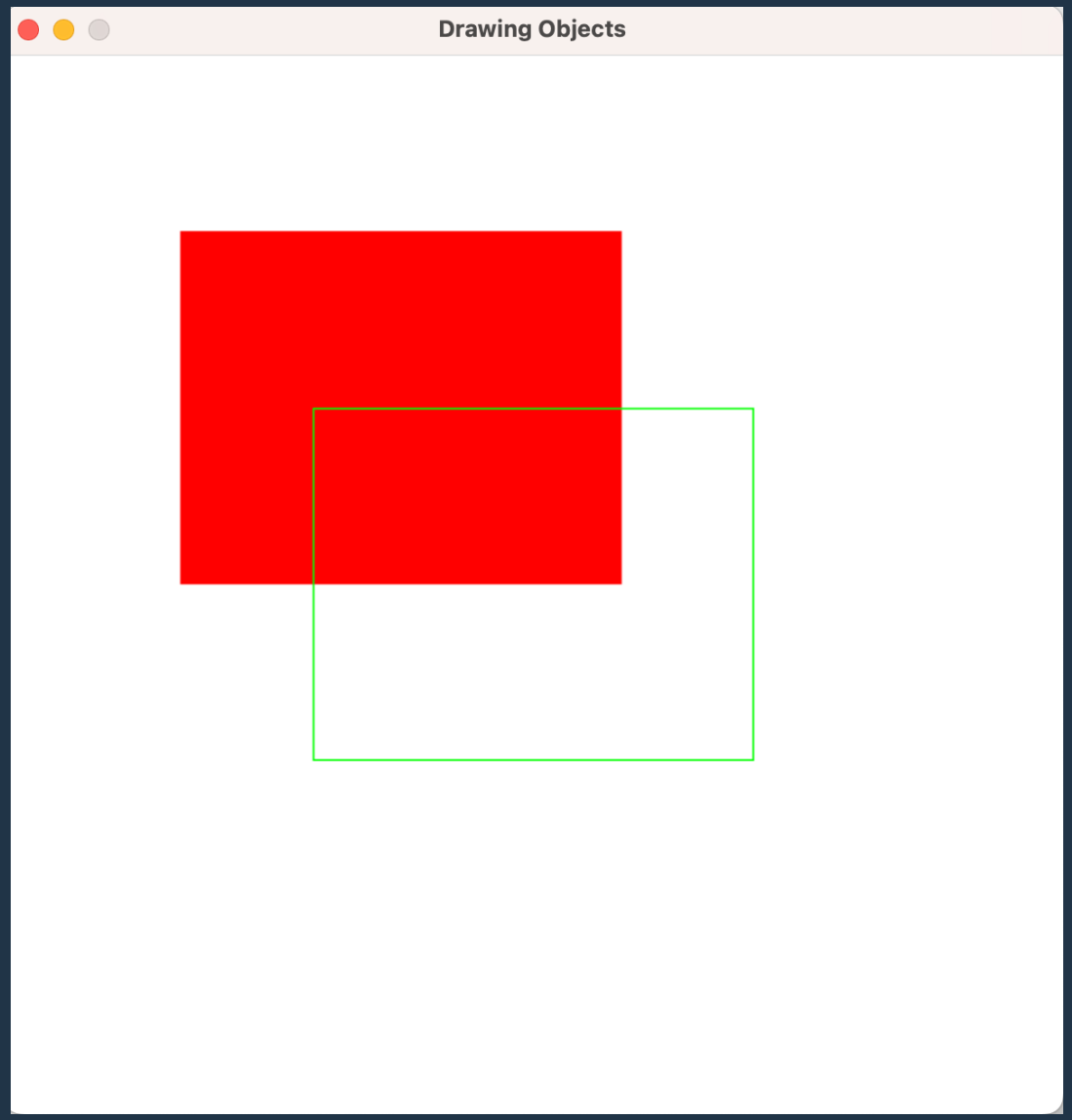
Let’s play with circles, lines and rectangles

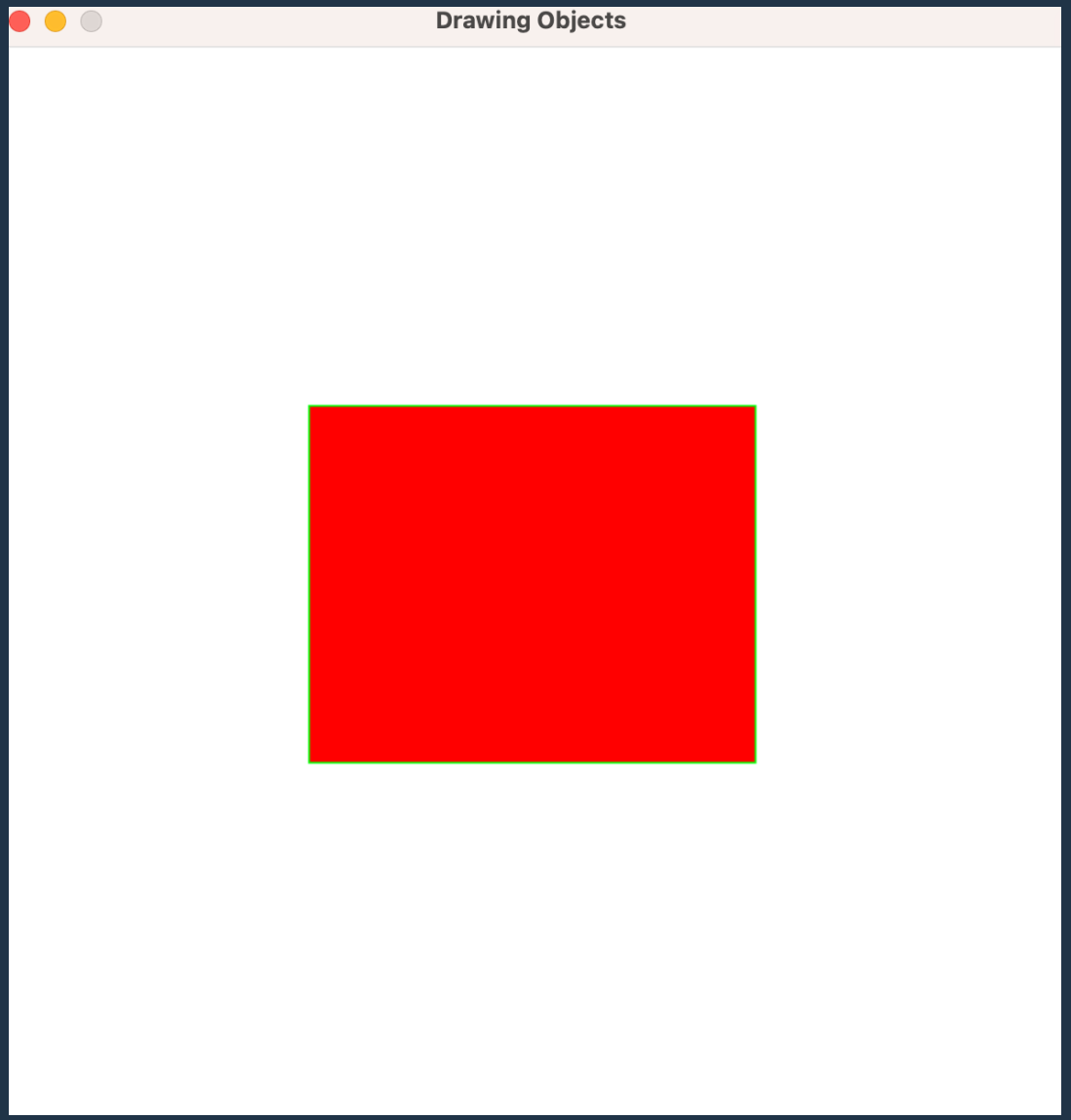
```
# Draw various shapes on our display
# Line(surface, color, starting point, ending point, thickness)
pygame.draw.line(display_surface, CYAN, (0, 0), (100, 100), 25)

# Circle(surface, color, center,  radius, thickness...0 for fill)
pygame.draw.circle(display_surface, WHITE, (WINDOW_WIDTH/2, WINDOW_HEIGTH/2), 100, 10)

# Rectangle(surface, color, (top-left x, top-left y, width, height))
pygame.draw.rect(display_surface, MAGNETA, (0, 0, 100, 100))
```

One little task for you!



Task 1: Place the rectangles exactly in the center of the screen





Answer:

```
WINDOW_WIDTH = 600
WINDOW_HEIGTH = 600
rect_width, rect_height = 250, 200
rect_x = WINDOW_WIDTH/2 - rect_width/2
rect_y = WINDOW_HEIGTH/2 - rect_height/2
pygame.draw.rect(display_surface, RED, (rect_x, rect_y, rect_width, rect_height))
```

Discrete keyboard movement

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_UP:
        rect_y = rect_y - STEP
    if event.key == pygame.K_DOWN:
        rect_y = rect_y + STEP
    if event.key == pygame.K_LEFT:
        rect_x -= STEP
    if event.key == pygame.K_RIGHT:
        rect_x += STEP
```

Pressing a key its an event in Pygame.

Define what key we pressed and what have to do right after pressing.

Continuous movement

Step 1: Let's define flags

```
# Define the flags
rect_is_moving_up = False
rect_is_moving_down = False
rect_is_moving_left = False
rect_is_moving_right = False
```

Step 2: Let's define the logic of changing the flag

```
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_LEFT :
        rect_is_moving_left = True
    if event.key == pygame.K_RIGHT :
        rect_is_moving_right = True
    if event.key == pygame.K_UP:
        rect_is_moving_up = True
    if event.key == pygame.K_DOWN:
        rect_is_moving_down = True
```

```
if event.type == pygame.KEYUP:
    if event.key == pygame.K_LEFT:
        rect_is_moving_left = False
    if event.key == pygame.K_RIGHT:
        rect_is_moving_right = False
    if event.key == pygame.K_UP:
        rect_is_moving_up = False
    if event.key == pygame.K_DOWN:
        rect_is_moving_down = False
```

Step 3: Let's try to move

Notice:

THE HOLDING OF KEY IS NOT EVENT IN Pygame

```
if rect_is_moving_left:
    rect_x -= STEP
if rect_is_moving_right:
    rect_x += STEP
if rect_is_moving_up:
    rect_y -= STEP
if rect_is_moving_down:
    rect_y += STEP
```

Continuous movement

Step 1: Lets the key being pressed

```
#Get a list of all keys currently being pressed down
keys = pygame.key.get_pressed()
print(keys)
```

Step 2: Let's try to move

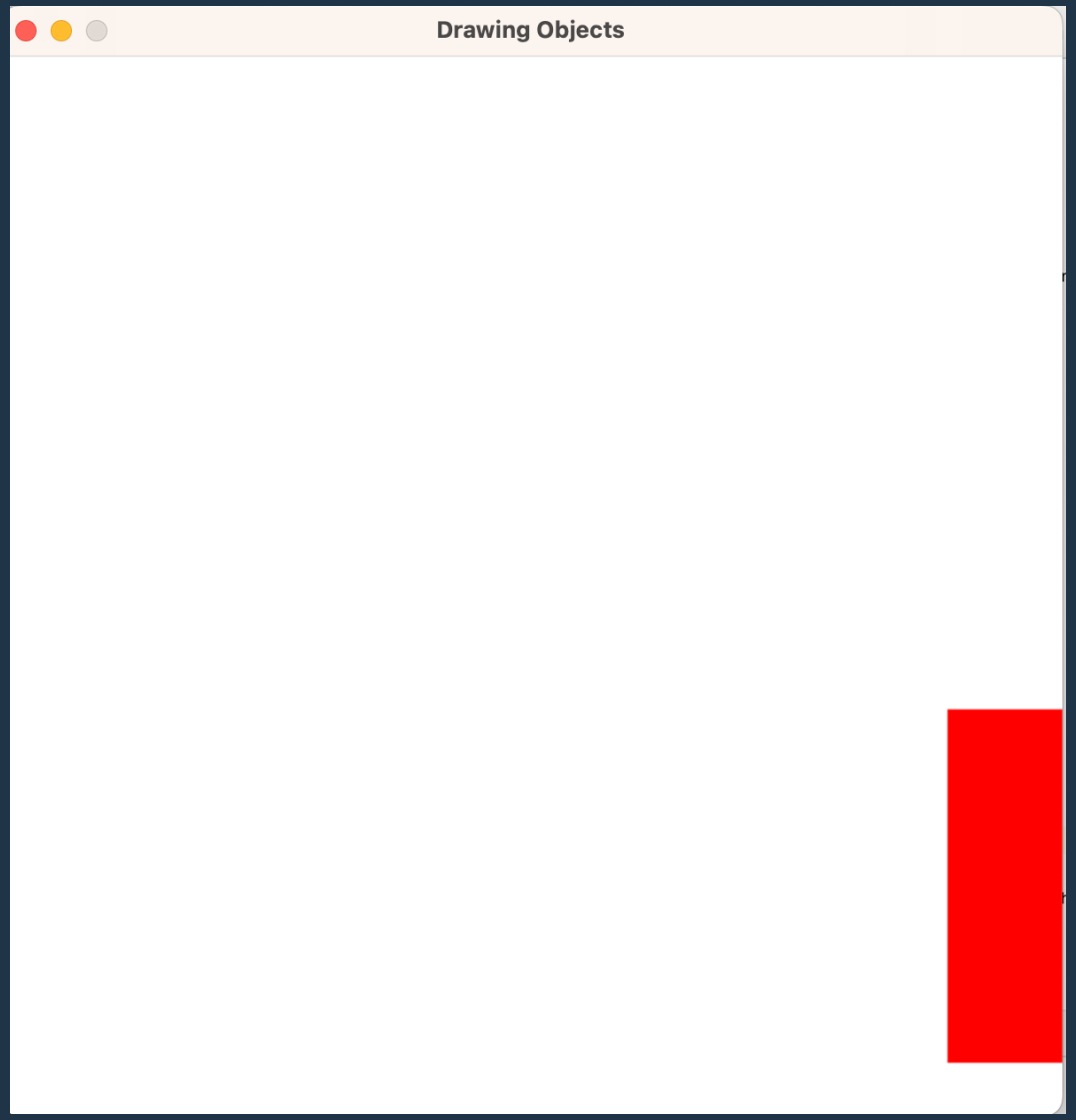
```
#Move continuously our rect
if keys[pygame.K_LEFT]:
    rect_x -= STEP
if keys[pygame.K_RIGHT]:
    rect_x += STEP
if keys[pygame.K_UP]:
    rect_y -= STEP
if keys[pygame.K_DOWN]:
    rect_y += STEP
```

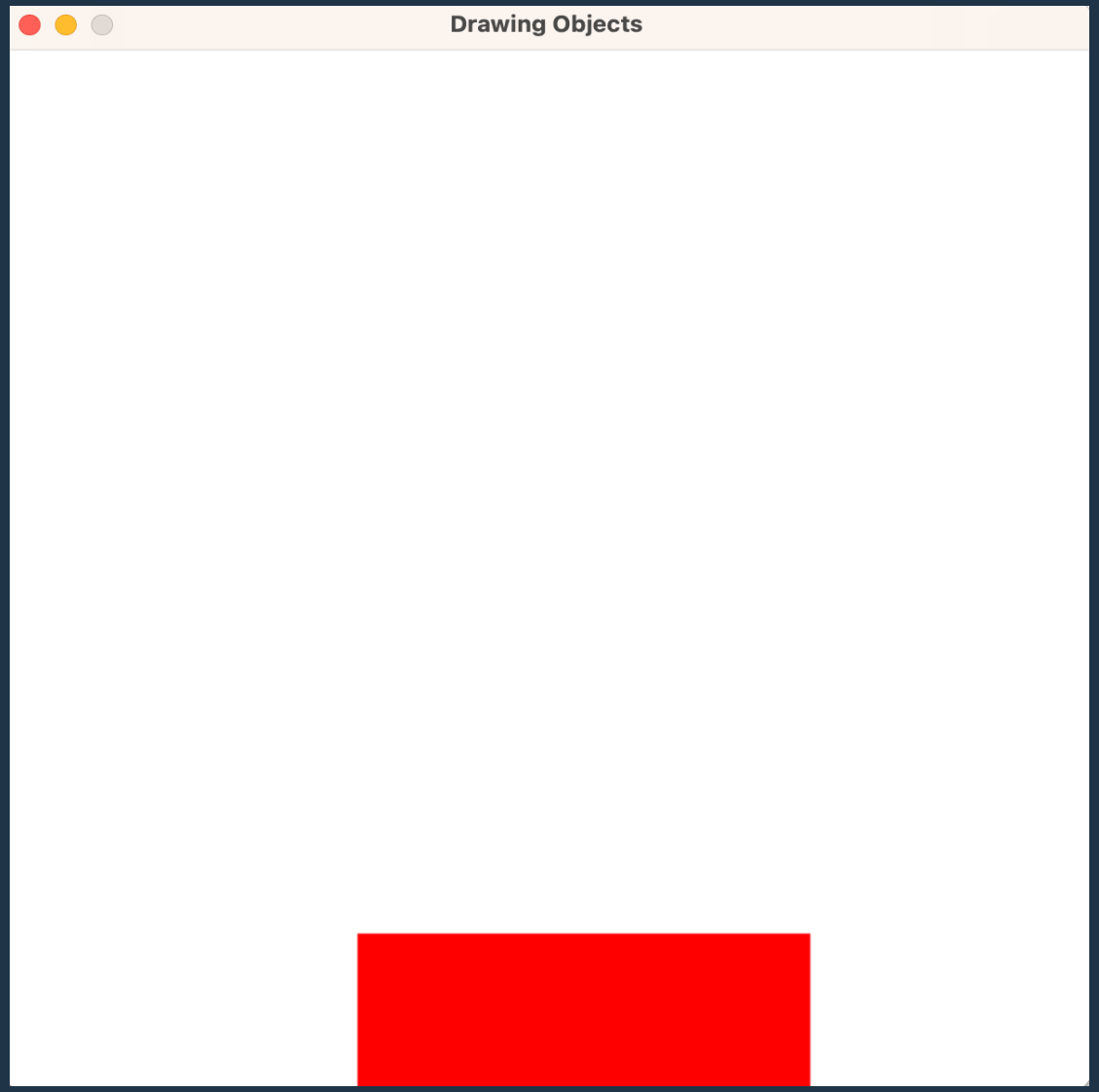
Notice:

THE HOLDING OF KEY IS NOT EVENT IN Pygame

One more task for you!

Task 2: Restrict the movement out of screen





Answer: How to restrict the movement of object out of screen frame

```
if rect_is_moving_left and rect_x >= STEP:
    rect_x -= STEP
if rect_is_moving_right and rect_x <= WINDOW_WIDTH - rect_width - STEP:
    rect_x += STEP
if rect_is_moving_up and rect_y >= STEP:
    rect_y -= STEP
if rect_is_moving_down and rect_y <= WINDOW_HEIGTH - rect_height - STEP:
    rect_y += STEP
```

Blitting images

Step 1:

```
# Create images, returns a surface objects with the image draw on it.
# We can then get the rect of the surface and use the rect to position the image.
dragon_image = pygame.image.load("icons/image.png")
dragon_rect = dragon_image.get_rect()
dragon_rect.topleft = (0, 0)
```

Step 2:

```
# The main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Blit a surface objects at the given cordinates to our display
    display_surface.blit(dragon_image, dragon_rect )

    # Update display
    pygame.display.update()

# End the game
pygame.quit()
```

Resource to download icons:
iconarchive.com

Blitting text: define fonts

Step 1:

```
# Available system fonts
fonts = pygame.font.get_fonts()
for font in fonts:
    print(font)
```

Step 2:

```
# Define fonts
system_font = pygame.font.SysFont('calibri', 64)
custom_font = pygame.font.Font('RedBlock.ttf', 32)
```

Resource to download fonts:
fontspace.com/

Blitting text:
Blit the text on the screen

Step 1:

```
# Define text
system_text = system_font.render("Dragon Rule!", True, GREEN, DARKGREEN)
system_text_rect = system_text.get_rect()
system_text_rect.center = (WINDOW_WIDTH//2, WINDOW_HEIGTH//2)
```

Step 2:

```
# The main game loop
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # Blit the text at the given cordinates to our display
    display_surface.blit(system_text, system_text_rect)
    display_surface.blit(custom_text, custom_text_rect)

    # Update display
    pygame.display.update()

# End the game
pygame.quit()
```

Add sound effects and music: define sounds effects, set volume

Step 1:

```
# Define sound
sound_1 = pygame.mixer.Sound('sound_1.wav')
sound_2 = pygame.mixer.Sound('sound_2.wav')
```

Step 2:

```
# play the sound effects
sound_1.play()
pygame.time.delay(2000)
sound_2.play()
```

```
# Change the volume of a sound effect
sound_2.set_volume(.1)
sound_2.play()
```

Resource to download sound effects:
leshylabs.com

Add sound effects and music: make a background music

Step 1:

```
#Load background music
pygame.mixer.music.load('music.wav')
```

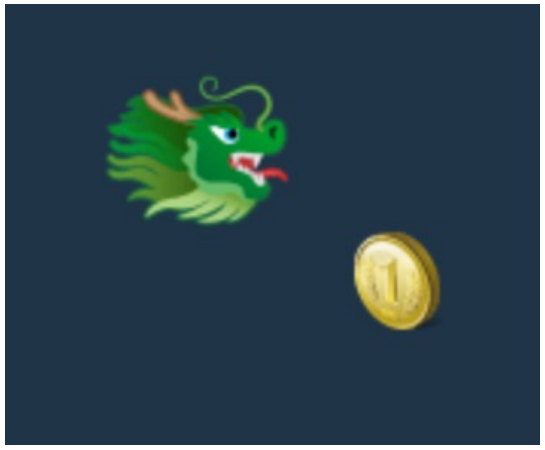
Step 2:

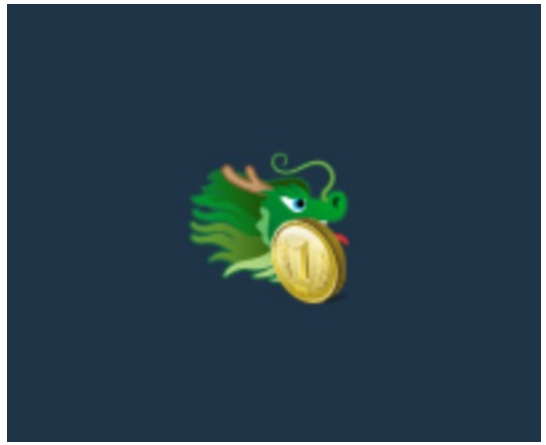
```
#Play and stop music
pygame.mixer.music.play(-1, 0.0)
pygame.time.delay(10000)
pygame.mixer.music.stop()
```

Resource to download sound effects:
leshylabs.com

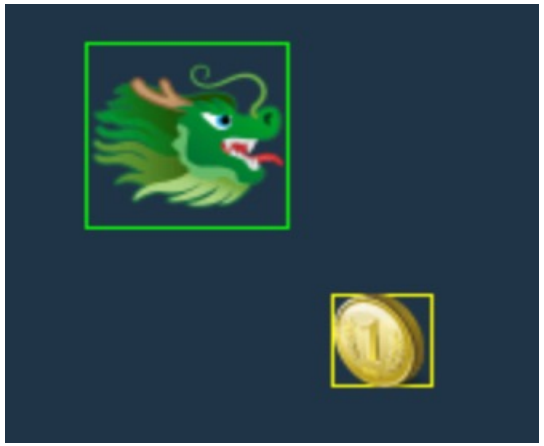
Collision Detection:

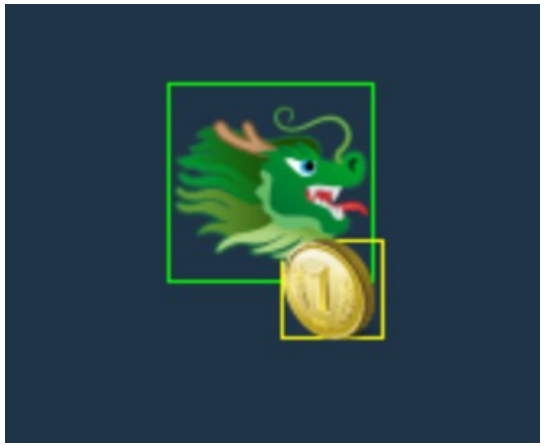
Let’s detect when the objects are touching each other:





Task 3: Lets draw rectangles to show the rect frame of our objects





Collision Detection:

Step 0: Lets draw rectangles to show the rect frame of our objects

```
# Draw rectangles to represent the rect's of each object
pygame.draw.rect(display_surface, (0, 255, 0), dragon_right_rect, 1)
pygame.draw.rect(display_surface, (255, 255, 0), coin_rect, 1)
```

Step 1: detect when the objects to collide each other and make some actions with them

```
#Check for collision between two rects
if dragon_right_rect.colliderect(coin_rect):
    print('HIT')
    coin_rect.x = random.randint(0, WINDOW_WIDTH - 32)
    coin_rect.y = random.randint(0, WINDOW_HEIGTH - 32)
```

One more crucial things!

FPS

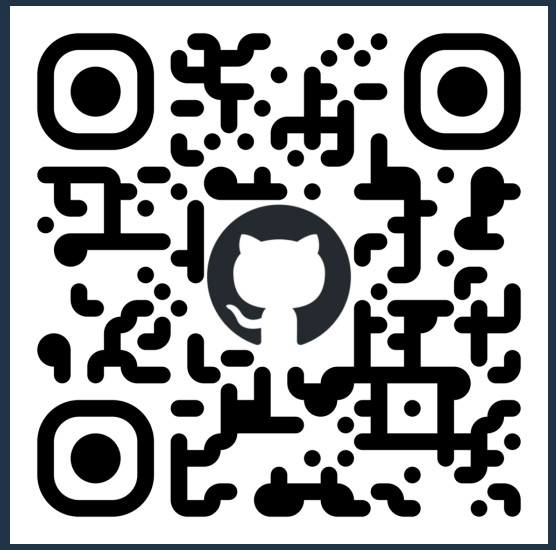
Conclusion

Congratulations, great game design is now within your reach!

Thanks to Python and a buffet of highly capable Python game engines, you can create your first computer game much more easily than before

Thank you!
Let’s keep in touch

https://www.linkedin.com/in/halloweex



https://github.com/halloweex