Game

Part 4

Complete class ---- square colour ANSWER

```
import pygame
...
square_colour = (255, 0, 100)
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
        self.colour = colour
        self.image.fill(colour)

done = False
while not done:
...
pygame.quit()
```

Complete class ---- square colour ANSWER

```
import pygame
...
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y

        self.colour = colour
        self.image.fill(colour)

my_square = Square(400 - 20/2 ,300 - 20/2,20,5,5,(100, 10, 255))
my_square2 = Square(400 - 20/2 ,100, 20,5,5,(100, 10, 255))

...
pygame.quit()
```

Let's test it!!

You can give the same colour for now. Of course you can give different colours and observe result.

The main purpose of testing is to quickly test something out. We are just checking if the constructor is working or not. Remember?

Complete class ---- square colour ANSWER

```
import pygame
...
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.image.fill(colour)
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y

        self.colour = colour
        self.image.fill(colour)

my_square = Square(400 - 20/2 ,300 - 20/2,20,5,5,(100, 10, 255))
my_square2 = Square(400 - 20/2 ,100, 20,5,5,(100, 10, 255))

...
```

Let's test it!!

You can give the same colour for now. Of course you can give different colours and observe result.

The main purpose of testing is to quickly test something out. We are just checking if the constructor is working or not. Remember?

square colour possible method

```
import pygame
...

class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.image.fill((random.randint(0, 255), 0, 0))
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
allspriteslist = pygame.sprite.Group()
for i in range(5):
    s = Square(random.randint(0, 800), random.randint(0, 600), 20, random.randint(-3,3), random.randint(-3,3))
    allspriteslist.add(s)
...
pygame.quit()
```

You might ask why can't we just give it a random colour inside the constructor.

The reason being is that we won't be able to change the class all the time. We try to write a class once and for all.

This approach always gives a random colour and there is no control during the instantiation (creation). You can always reassign a new colour after the instantiation, but why not? It is a waste of time if you can do it from the beginning

square colour possible method

```
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.image.fill(colour)
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
...
allspriteslist = pygame.sprite.Group()
s2 = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3),
random.randint(-3,3),(200,0,0))
allspriteslist.add(s2)
for i in range(5):
    s = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3),
random.randint(-3,3), (random.randint(0, 255), 0, 0))
    allspriteslist.add(s)
...
pygame.quit()
```

This is a better solution. We will keep the class to use whatever colour we give. Simple and clean.

How we use it is the tricky part. We can either choose a fixed color, or a completely randomized colour when we create the object.

Create an object with unchanged colour

One line !

square colour 2

increase the amount of squares

```
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.image.fill(colour)
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
...
allspriteslist = pygame.sprite.Group()
s2 = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3), random.randint(-3,3),
(200,0,0))
allspriteslist.add(s2)
for i in range(5):
    s = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3), random.randint(-3,3),
(random.randint(0, 255), 0, 0))
    allspriteslist.add(s)
...
pygame.quit()
```

One line !

square colour 2

```
class Square(pygame.sprite.Sprite):
    def __init__(self, x, y, side, speed_x, speed_y, colour):
        super().__init__()
        self.side = side
        self.image = pygame.Surface([side, side])
        self.speed_x=speed_x
        self.speed_y=speed_y
        self.image.fill(colour)
        self.rect=self.image.get_rect()
        self.rect.x=x
        self.rect.y=y
...

allspriteslist = pygame.sprite.Group()
for i in range(5):
    s = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3),
random.randint(-3,3), (random.randint(0, 255), 0, 0))
    allspriteslist.add(s)

...
pygame.quit()
```

Now a little challenge

Display only 1 Square without using the for-loop to create Square

(hint: create ONE Square, and add it to the allspriteslist)

increase the size of Square, try doing this without modifying the class

(hint: where did we say the size would be?)

Without any speed

(hint: think about how the square got its position updated, there is two solutions)

Solution

```
import pygame
...

allspriteslist = pygame.sprite.Group()
s = Square(random.randint(0, 800), random.randint(0, 600), 59, 0, 0, (200, 0, 0))
allspriteslist.add(s)

...
pygame.quit()
```

More event -- MOUSEEVENT

There is a function **pygame.mouse.get_pos().**
It returns the position of your mouse.
You can call it most of the time. Let’s give it a try.

Let’s add this line of code in our code. Suggest me where to put it.
print(pygame.mouse.get_pos())

More event -- MOUSEEVENT

Calling this function all the time is going be expensive on computer resources.

What if I want to only get a position when I click my mouse.

We can try to capture mouse event in our game loop. Those are the times we need the position of the mouse.

Position of MOUSE -- MOUSEBUTTONDOWN:

```
while not done:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
          pos = pygame.mouse.get_pos()

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                done = True
            if event.key == pygame.K_b:
                background_colour = (0,0,200)
            if event.key == pygame.K_g:
                background_colour = (0,200,0)

    screen.fill(background_colour)
    allspriteslist.draw(screen)
    pygame.display.flip()
    allspriteslist.update()

    clock.tick(120)
pygame.quit()
```

MOUSEBUTTONUP ?

For all the event occur during our game, we use this for-loop to keep track of them. If there is an event of MOUSEBUTTONDOWN, we will store it first.

Remove the sprite from group

```
while not done:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
            s.remove(allspriteslist)

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                done = True
            if event.key == pygame.K_b:
                background_colour = (0,0,200)
            if event.key == pygame.K_g:
                background_colour = (0,200,0)

    screen.fill(background_colour)
    allspriteslist.draw(screen)
    pygame.display.flip()
    allspriteslist.update()
...
pygame.quit()
```

We can try another remove. Note that we don't have to click on the Square to remove it at the moment. It has nothing to do with mouse position yet.

Remove the sprite from group

```
while not done:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
                if s.rect.collidepoint(pos):
                    s.remove(allspriteslist)

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                done = True
            if event.key == pygame.K_b:
                background_colour = (0,0,200)
            if event.key == pygame.K_g:
                background_colour = (0,200,0)

    screen.fill(background_colour)
    allspriteslist.draw(screen)
    pygame.display.flip()
    allspriteslist.update()
...
pygame.quit()
```

Now let's connect them with this function. It is the rect object function called collidepoint(). It returns true or false, depending on whether the mouse position was within the rect object.

Remember? The rect object inside our Square is to used for motion and location. So it also know its size, and has a function to let you know whether a point is within it.

Remove the sprite from group ——

sprite.rect.collidepoint(pos):

We only do checking on one single Squares. And now let's put all the Squares back in the allspriteslist. So instead of using the Square objects with one single variable, we want to check with the all the Square objects through allspriteslist. Although allspriteslist is a Group object, it can be iterated (going through everything inside) just like a list!!

Remove the sprite from group

```
while not done:
    for event in pygame.event.get():
        if event.type == pygame.MOUSEBUTTONDOWN:
            pos = pygame.mouse.get_pos()
            for sprite in allspriteslist:
                if sprite.rect.collidepoint(pos):
                    sprite.remove(allspriteslist)

        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_q:
                done = True
            if event.key == pygame.K_b:
                background_colour = (0,0,200)
            if event.key == pygame.K_g:
                background_colour = (0,200,0)

...
pygame.quit()
```

Create a for-each loop and a temp variable temp, to represent all sprite object from time to time, and one at a time.

Apply the same logic we used when dealing with one variable.

More Squares

```
allspriteslist = pygame.sprite.Group()

for i in range(10):
    s = Square(random.randint(0, 800), random.randint(0, 600), 59, random.randint(-3,3), random.randint(-3,3), (255, 0, 0))
    allspriteslist.add(s)
```