Squash in Python

2021 年 2 月 23 日

S.Matoike

目次

第 1 章

はじめに

第 2 章

スカッシュ（Squash）ゲーム

2.1　Pygame

2.1.1　Window の表示

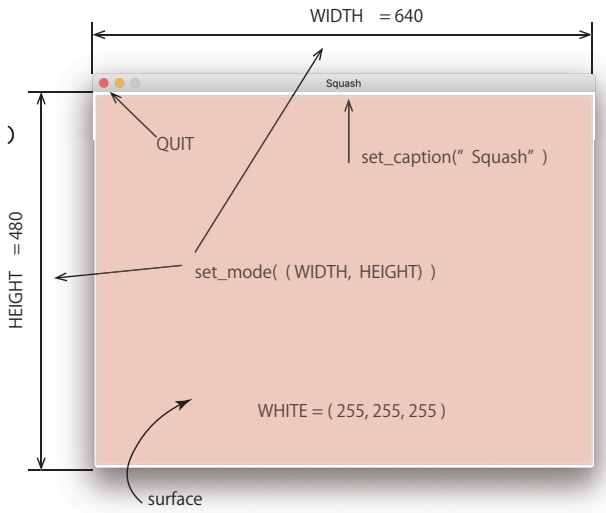
Window の surface サイズは、横幅 WIDTH と高さ HEIGHT のタプル (WIDTH, HEIGHT) で指定します。ここでは、このタプルを WSIZE という名前の変数に納めて使っています

Window のキャプションを文字列で指定します

キーボード・イベントの取得方法を確認しましょう

ソースコード 2.1　Window を用意する

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4
5  pygame.init()
6  WIDTH = 640
7  HEIGHT = 480
8  WSIZE = (WIDTH, HEIGHT)
9  surface = pygame.display.set_mode( WSIZE )
10 pygame.display.set_caption( 'Squash' )
11 clock = pygame.time.Clock()
12 FPS = 10
13 WHITE = (255,255,255)
14
15 while True:
16     for event in pygame.event.get():
17         if event.type == QUIT:
18             pygame.quit()
19             sys.exit()
20     surface.fill( WHITE )
21     pygame.display.update()
22     clock.tick( FPS )
```



【演習】

(1) surface のサイズを変更して実行してみましょう

(2) pygame.display.set_mode() の第 2 引数に FULLSCREEN を指定してみましょう

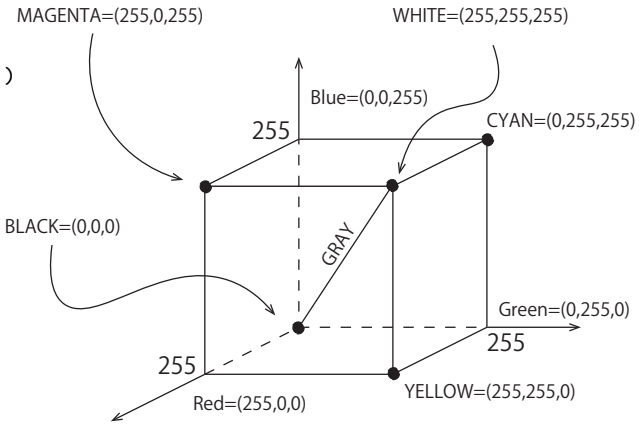
(3) キャプションを変えてみましょう

2.1.2　色の指定と画面の更新

色の定義は、Red、Green、Blue のそれぞれを 256 段階 (0〜255) で指定したタプルです
1 秒あたりの画面更新回数を整数値 FPS（Frames Per Second）で指定（1≦FPS≦30 程度）します
pygame.display.update() は、引数を指定しない場合、画面全体を更新します

ソースコード 2.2　塗りつぶしの色を指定する

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4  import random
5
6  pygame.init()
7  WIDTH = 640
8  HEIGHT = 480
9  WSIZE = (WIDTH, HEIGHT)
10 surface = pygame.display.set_mode( WSIZE )
11 pygame.display.set_caption( 'Squash' )
12 clock = pygame.time.Clock()
13 FPS = 2
14
15 RED = (255,0,0)
16 GREEN = (0,255,0)
17 BLUE = (0,0,255)
18 YELLOW = (255,255,0)
19 MAGENTA = (255,0,255)
20 CYAN = (0,255,255)
21 WHITE = (255,255,255)
22 BLACK = (0,0,0)
23 COLORS = [RED, GREEN, BLUE,\
24           YELLOW, CYAN, MAGENTA, WHITE, BLACK]
25
26 while True:
27     for event in pygame.event.get():
28         if event.type == QUIT:
29             pygame.quit()
30             sys.exit()
31     n = random.randint( 0, 7 )
32     surface.fill( COLORS[n] )
33     pygame.display.update()
34     clock.tick( FPS )
```



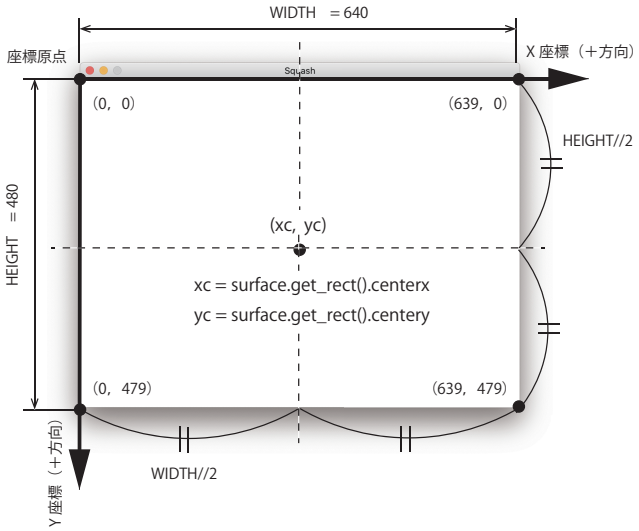
256 X 256 X 256 ＝ 16777216 ＝ 約千六百七十万色
Full Color（フル・カラー）

n = random.randint(0, 7) は、0≦n≦7 の範囲の整数をランダムに生成しています
確認のために、print(n)　を一行入れてみたら理解の助けになるかもしれません

【演習】

(1) 白と黒の場合の（Red,Green,Blue）の値を、実際に設定して確認してみよう
(2) Red,Green,Blue に同じ値を指定すると、灰色の濃さが変わることを確認しよう
(3) 自分で好みの色を作って screen を塗りつぶしてみよう
(4) FPS の値を変えて実行してみよう
(5) FPS の値を大きくしすぎると、どんな問題を生じる可能性があるか考えてみよう

2.1.3　各種図形の描画



各種図形を描くときの、共通の引数として surface と color があります

- surface は、その図形を描画する盤面（pygame.display.set_mode で生成した）
- 色の指定 color は、Red,Green,Blue のタプル

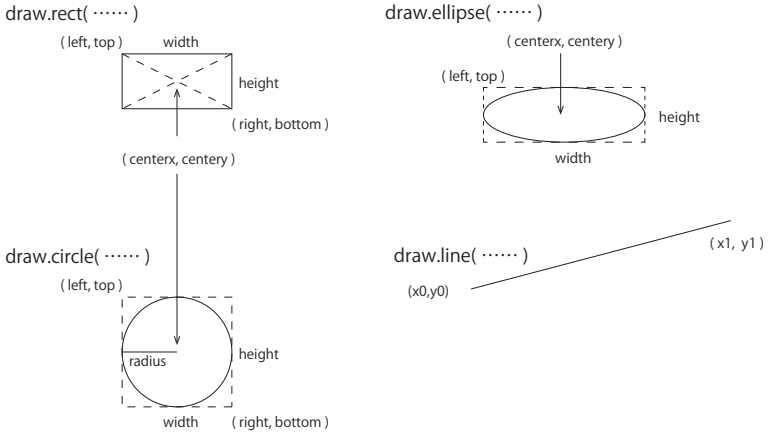
以下では、個々の図形のそのほかの引数について説明します

(1) 円：pygame.draw.circle(Surface, color, position, radius, width=0)
  - 円の中心位置 position は、X 座標と Y 座標のタプル
  - 円の半径 radius は整数値

(2) 線：pygame.draw.line(Surface, color, start_position, end_position, width=1)
  - 始点 start_position と終点 end_position は、各々 X 座標と Y 座標のタプル
  - 線の幅 width=1 はデフォルト引数 (=1 なら省略可能)

(3) 矩形：pygame.draw.rect(Surface, color, Rect, width=0)
  - 線の幅 width=0 はデフォルト引数 (=0 なら省略可能)
  - 矩形を納める Rect は、left, top, width, height のタプル

(4) 楕円：pygame.draw.ellipse(Surface, color, Rect, width=0)
  - 楕円が収まる Rect を、left, top, width, height のタプルで指定

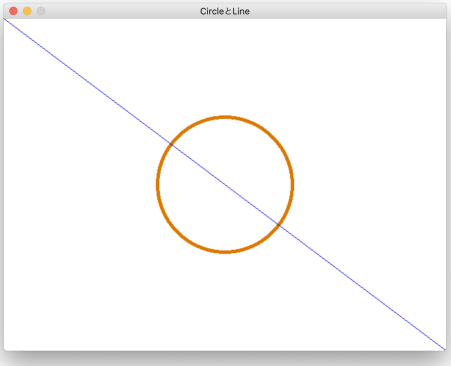


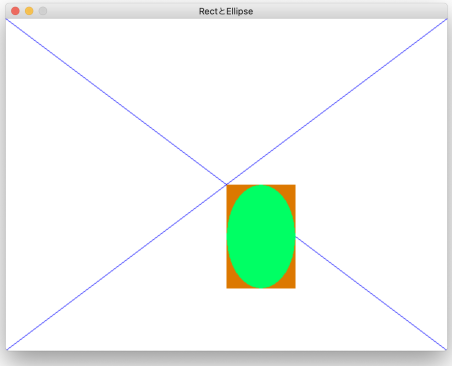
次の作図を試してみましょう

- surface の中央（ WIDTH//2, HEIGHT//2）に、半径 radius（=100 ピクセル）の円 Circle
- surface の左上（ left=0, top=0 ）を始点、右下（ right=WIDTH-1, bottom=HEIGHT-1 ）を終点とする線 Line

ソースコード 2.3　Circle と Line

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4
5  pygame.init()
6  WIDTH = 640
7  HEIGHT = 480
8  WSIZE = (WIDTH, HEIGHT)
9  surface = pygame.display.set_mode( WSIZE )
10 pygame.display.set_caption( 'CircleとLine' )
11 clock = pygame.time.Clock()
12 FPS = 1
13
14 WHITE = (255,255,255)
15 r = 220
16 g = 120
17 b = 0
18 xc = WIDTH//2
19 yc = HEIGHT//2
20 radius = 100
21 left = top = 0
22 right = WIDTH - 1
23 bottom = HEIGHT -1
24
25 while True:
26     for event in pygame.event.get():
27         if event.type == QUIT:
28             pygame.quit()
29             sys.exit()
30     surface.fill( WHITE )
31     pygame.draw.circle( surface, (r, g, b), (xc, yc), radius, width=5 )
32     pygame.draw.line( surface, (0, 0, 255), (left, top), (right, bottom) )
33     pygame.display.update()
34     clock.tick( FPS )
```





次の作図を試してみましょう

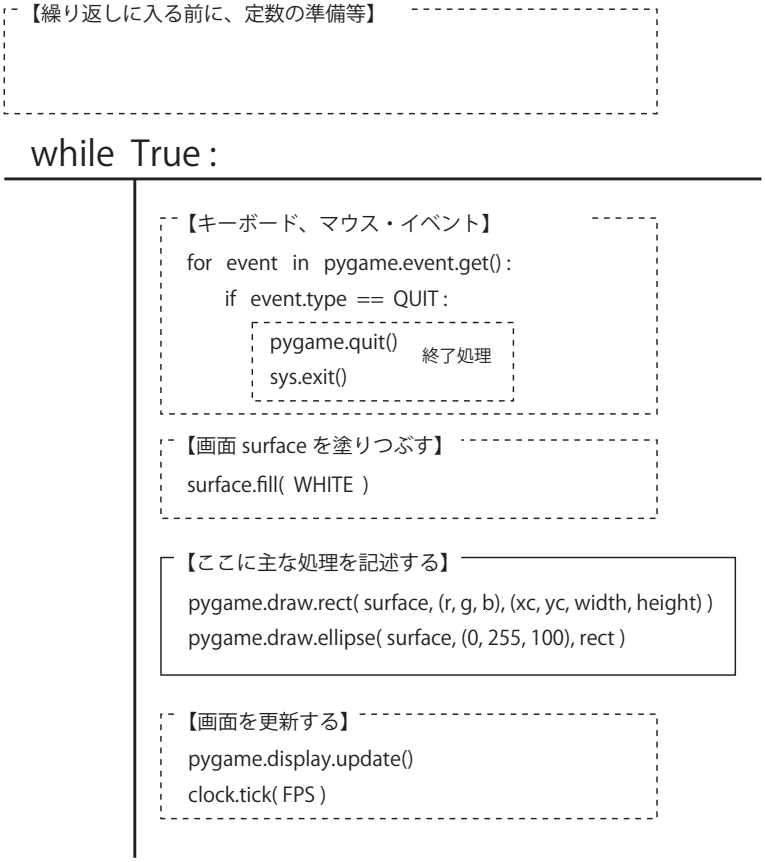
- surface の中央 (WIDTH//2, HEIGHT//2) を四角形 Rect の左上の点 (left, top) とし、幅 width を 100、高さ height を 150 の Rect
- 上記の Rect に収まる楕円 Ellipse

ソースコード 2.4　Rect と Ellipse

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, Rect
4
5  pygame.init()
6  WIDTH = 640
7  HEIGHT = 480
8  WSIZE = (WIDTH, HEIGHT)
9  surface = pygame.display.set_mode( WSIZE )
10 pygame.display.set_caption( 'RectとEllipse' )
11 clock = pygame.time.Clock()
12 FPS = 1
13
14 WHITE = (255,255,255)
15 r = 220
16 g = 120
17 b = 0
18 xc = WIDTH//2
19 yc = HEIGHT//2
20 width = 100
21 height = 150
22 rect = Rect(xc, yc, width, height)
23
24 while True:
25     for event in pygame.event.get():
26         if event.type == QUIT:
27             pygame.quit()
28             sys.exit()
29     surface.fill( WHITE )
30     pygame.draw.line( surface, (0, 0, 255), (0, 0), (639, 479) )
31     pygame.draw.line( surface, (0, 0, 255), (639, 0), (0, 479) )
32     pygame.draw.rect( surface, (r, g, b), (xc, yc, width, height) )
33     pygame.draw.ellipse( surface, (0, 255, 100), rect )
34     pygame.display.update()
35     clock.tick( FPS )
```

【演習】random を import して、次のことを試してみましょう (from random import randint)

(1) circle の中心の x 座標を randint(0, WIDTH-1) で、y 座標も同様にして作図してみよう
(2) circle の半径を、乱数 randint(1,60) 程度で発生させ、中心座標と共に変化させてみよう
(3) 乱数で r=randint(0,255)、g、b を作り、タプル (r,g,b) で circle の色を指定してみまよう
(4) ellipse や rect あるいは line などを、乱数を使って surface 上に表示させてみよう
(5) surface.fill() をコメントアウトしてみよう



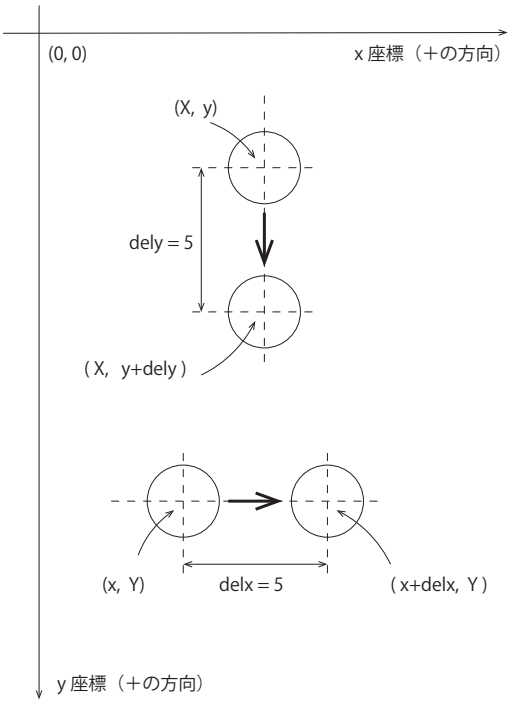
2.1.4　図形の移動

circle で描いたボールの y 座標を、フレーム更新のたびに増加させて、ボールを動かします
ボールの y 座標が、surface の HEIGHT に達したら終わります
QUIT をイベントとして検出した際に実行される終了処理を、fine() 関数にまとめています
surface.get_rect().width によって得られるのは、WIDTH
surface.get_rect().height によって得られるのは、HEIGHT

ソースコード 2.5　図形の移動

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4
5  def fine():
6      pygame.quit()
7      sys.exit()
8
9  pygame.init()
10 WIDTH = 640
11 HEIGHT = 480
12 WSIZE = (WIDTH, HEIGHT)
13 surface = pygame.display.set_mode( WSIZE )
14 pygame.display.set_caption( 'Squash (Move Ball)' )
15 clock = pygame.time.Clock()
16 FPS = 10
17
```

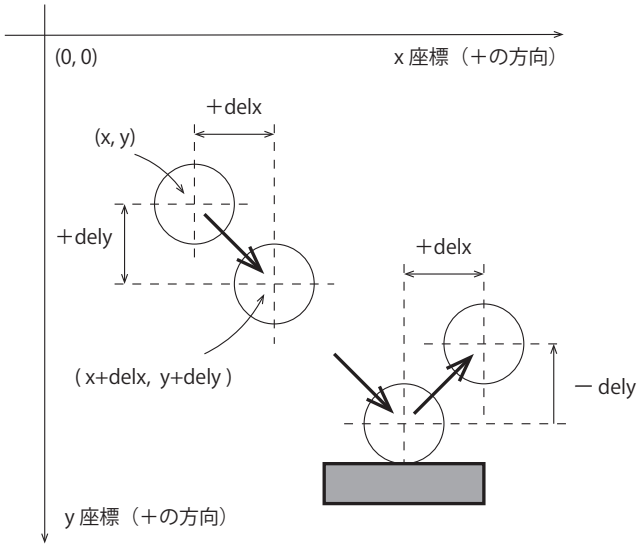
```
18 WHITE = (255,255,255)
19 RED = (255,0,0)
20 RADIUS = 10
21 x = surface.get_rect().centerx
22 y = 0
23 while True:
24     for event in pygame.event.get():
25         if event.type == QUIT:
26             fine()
27     surface.fill( WHITE )
28     pygame.draw.circle( surface, RED, (x,y), RADIUS )
29     if y>=HEIGHT:
30         break
31     y += 5
32     pygame.display.update()
33     clock.tick( FPS )
34
35 fine()
```



【演習】

(1) FPS を変えずにボールの移動するスピードを変えるには、どうすればいいでしょう
(2) ボールが、画面中央を下から上に動くようにプログラムを書き直してみましょう
(3) ボールが、画面中央を左から右に動くようにプログラムを書き直してみましょう
(4) ボールが、画面中央を右から左に動くようにプログラムを書き直してみましょう
(5) ボールが、画面の左上から右下に向かって動くようにプログラムを書き直してみましょう
(6) ボールが、画面の左下から右上に向かって動くようにプログラムを書き直してみましょう
(7) x 方向座標値の増分と y 方向座標値の増分の値が等しくない様にしたら、ボールの動きはどうなりますか

2.1.5　壁で反射するボール



circle で描いたボールの y 座標を、フレーム更新のたびに増加させて、ボールを動かします
ボールの y 座標が、surface の HEIGHT に達したら、y 方向の移動量をマイナスの値にしています
ボールの y 座標が、surface の上端に達したら、y 方向の移動量をプラスの値に戻しています

ソースコード 2.6　反射：位置変化量の符号を反転

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4
5  def fine():
6      pygame.quit()
7      sys.exit()
8
9  pygame.init()
10 WIDTH = 640
11 HEIGHT = 480
12 WSIZE = (WIDTH, HEIGHT)
13 surface = pygame.display.set_mode( WSIZE )
14 pygame.display.set_caption( 'Squash(Bound Ball)' )
15 clock = pygame.time.Clock()
16 FPS = 10
17
18 WHITE = (255,255,255)
19 RED = (255,0,0)
20 RADIUS = 10
21 x = surface.get_rect().centerx
22 y = 0
23 dely = 5
24
25 while True:
26     for event in pygame.event.get():
27         if event.type == QUIT:
28             fine()
29
30     surface.fill( WHITE )
```

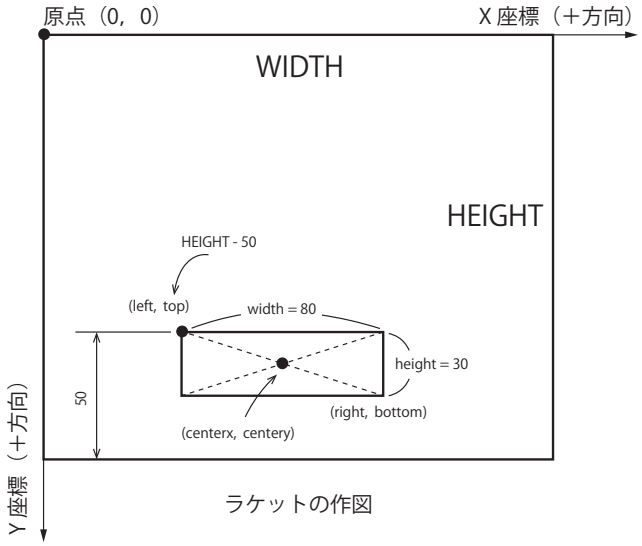
```
31      pygame.draw.circle( surface, RED, (x,y), RADIUS )
32      if y>=HEIGHT or y<0:
33          dely = -dely
34      y += dely
35      pygame.display.update()
36      clock.tick( FPS )
```

ボールが上下 y 方向で反射するように、y 方向の移動量の符号を反転しているのは前の例と同じです
ボールを左右 x 方向でも反射するように、x 方向の移動量の符号も反転させています
最初にボールを放出する位置 x 座標を、乱数で決めています

ソースコード 2.7　反射

```
1   import sys
2   import pygame
3   from pygame.locals import QUIT
4   import random
5
6   def fine():
7       pygame.quit()
8       sys.exit()
9
10  pygame.init()
11  WIDTH = 640
12  HEIGHT = 480
13  WSIZE = (WIDTH, HEIGHT)
14  surface = pygame.display.set_mode( WSIZE )
15  pygame.display.set_caption( 'Squash(Bounding Ball)' )
16  clock = pygame.time.Clock()
17  FPS = 30
18
19  WHITE = (255,255,255)
20  RED = (255,0,0)
21  RADIUS = 10
22  x = random.randint( 0, WIDTH-1 )
23  y = 0
24  delx = dely = 5
25
26  while True:
27      for event in pygame.event.get():
28          if event.type == QUIT:
29              fine()
30
31      surface.fill( WHITE )
32      pygame.draw.circle( surface, RED, (x,y), RADIUS )
33      if y>=HEIGHT or y<0:
34          dely = -dely
35      if x>=WIDTH or x<0:
36          delx = -delx
37      x += delx
38      y += dely
39      pygame.display.update()
40      clock.tick( FPS )
```

2.1.6　ラケットの導入



ラケットの作図

ラケットを pygame の Rect（クラスのオブジェクト）として作図し、それを左右の矢印キーを使って動かせるようにします

Rect クラスのオブジェクトは、次の 3 つの方法のどれかを使って生成できます

- pygame.Rect(left, top, width, height)
- pygame.Rect((left, top), (width, height))
- pygame.Rect(Rect オブジェクト)

Rect クラスは次のようなプロパティを持っており、値を設定することができます

top, left, bottom, right, topleft, bottomleft, topright, bottomright,

midtop, midleft, midbottom, midright, center, centerx, centery, size, width. height, w, h

このプログラム例では、ラケットの初期位置をラケットの Rect の左肩（left,top）について、画面の中央の縦線を left に、画面の下端から 50 ピクセル上を top に、重ねています

また、キーボードからのイヴェントを拾う key_event() 関数を新たに定義して、ゲームのメインループの記述を単純にしました

ソースコード 2.8　ラケットの Rect を描画

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, Rect
4  import random
5
6  def fine():
7      pygame.quit()
8      sys.exit()
9
10 def key_event():
11     for event in pygame.event.get():
12         if event.type == QUIT:
13             fine()
14
```

```
15 pygame.init()
16 WIDTH = 640
17 HEIGHT = 480
18 WSIZE = (WIDTH, HEIGHT)
19 surface = pygame.display.set_mode( WSIZE )
20 pygame.display.set_caption( 'Squash(Bounding Ball)' )
21 clock = pygame.time.Clock()
22 FPS = 20
23
24 WHITE = (255,255,255)
25 # ボールのdraw.circleのために
26 RED = (255,0,0)
27 RADIUS = 10
28 x = random.randint( 0, WIDTH-1 )
29 y = 0
30 delx = dely = 5
31 # ラケットのdraw.rectのために
32 GREEN = (0,255,0)
33 SIZE = (WIDTH//2, HEIGHT-50, 80, 10)
34 racket = Rect( SIZE )
35
36 while True:
37     key_event()
38     surface.fill( WHITE )
39     pygame.draw.rect( surface, GREEN, racket )
40     pygame.draw.circle( surface, RED, (x,y), RADIUS )
41     # 上下の壁でボールの反射
42     if y>=HEIGHT or y<0:
43         dely = -dely
44     # 上下の壁でボールの反射
45     if x>=WIDTH or x<0:
46         delx = -delx
47     # ボールの中心座標を更新
48     x += delx
49     y += dely
50     pygame.display.update()
51     clock.tick( FPS )
```

size や width 及び height の属性の変更は、四角形の大きさを変更することになり、それ以外のプロパティの変更は、四角形の大きさを変えずに、描画位置を変更することになります

このプログラム例では、ラケットの Rect の centerx プロパティを操作しています

キーボードの左右の矢印キーのイベントを拾って、右矢印（K_RIGHT）の時は racket.centerx+=15、左矢印（K_LEFT）の時は racket.centerx-=15 のように変えることによって、描画位置を変えています

ソースコード 2.9　ラケット

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT,KEYDOWN,K_RIGHT,K_LEFT,Rect
4  import random
5
6  def fine():
7      pygame.quit()
8      sys.exit()
9
10 def key_event():
```

```
11      for event in pygame.event.get():
12          if event.type == QUIT:
13              fine()
14          elif event.type == KEYDOWN:
15              if event.key == K_RIGHT:
16                  racket.centerx += 15
17              if event.key == K_LEFT:
18                  racket.centerx -= 15
19
20  pygame.init()
21  WIDTH = 640
22  HEIGHT = 480
23  WSIZE = (WIDTH, HEIGHT)
24  surface = pygame.display.set_mode( WSIZE )
25  pygame.display.set_caption( 'Squash(Bounding Ball)' )
26  clock = pygame.time.Clock()
27  FPS = 20
28
29  WHITE = (255,255,255)
30  RED = (255,0,0)
31  RADIUS = 10
32  x = random.randint( 0, WIDTH-1 )
33  y = 0
34  delx = dely = 5
35  GREEN = (0,255,0)
36  SIZE = (WIDTH//2, HEIGHT-50, 80, 10)
37  racket = Rect( SIZE )
38
39  while True:
40      key_event()
41      surface.fill( WHITE )
42      pygame.draw.rect( surface, GREEN, racket )
43      pygame.draw.circle( surface, RED, (x,y), RADIUS )
44      if y>=HEIGHT or y<0:
45          dely = -dely
46      if x>=WIDTH or x<0:
47          delx = -delx
48      x += delx
49      y += dely
50      pygame.display.update()
51      clock.tick( FPS )
```

2.1.7　ラケットとボールの干渉

ボールの位置は while True:のループの中でその都度動いていますから、毎回ボールの Rect を定義し直しては、ラケットの Rect と ボールの Rect との間に重なりが生じたか否かを、colliderect() メソッドを使って検出しています

左右方向の矢印キーを押した時、ラケットの動きをスクリーンの幅の中に制限 (ラケットの Rect の右側が画面の WIDTH 未満であることや、ラケットの Rect の左側が画面の左端=0 より大きい範囲に制限) する様にしています

ソースコード 2.10　ラケットとボールの干渉検出

```
import sys
import pygame
from pygame.locals import QUIT,Rect,KEYDOWN,K_RIGHT,K_LEFT
import random

def fine():
    pygame.quit()
    sys.exit()

def key_event():
    for event in pygame.event.get():
        if event.type == QUIT:
            fine()
        elif event.type == KEYDOWN:
            if event.key == K_RIGHT and racket.right < WIDTH:
                racket.centerx += 15
            if event.key == K_LEFT and racket.left > 0:
                racket.centerx -= 15

pygame.init()
WIDTH = 640
HEIGHT = 480
WSIZE = (WIDTH, HEIGHT)
surface = pygame.display.set_mode( WSIZE )
pygame.display.set_caption( 'Squash' )
clock = pygame.time.Clock()
FPS = 20

WHITE = (255,255,255)
RED = (255,0,0)
RADIUS = 10
x = random.randint( 0, WIDTH-1 )
y = 0
delx = dely = 5
GREEN = (0,255,0)
SIZE = (WIDTH//2, HEIGHT-50, 80, 10)
racket = Rect( SIZE )

while True:
    key_event()
    surface.fill( WHITE )
    pygame.draw.rect(surface, GREEN, racket)
    pygame.draw.circle( surface, RED, (x,y), RADIUS )
    ball_rect = Rect( (x-RADIUS, y-RADIUS, RADIUS*2, RADIUS*2) )
    if y<0 or racket.colliderect( ball_rect ):
        dely = -dely
    if x<0 or x>=WIDTH:
        delx = -delx
    if y>=HEIGHT:
        print( 'Game over' )
        break
    x += delx
    y += dely
    pygame.display.update()
    clock.tick( FPS )

```

```
57 fine()
```

ボールを打ち返し損じた場合、Game Over と print して終了（while のループを break で抜けて fine() 処理に入る）するようにしています

2.1.8　文字テキストの表示

「Game Over !」の文字テキストを、ゲームっぽく大きな文字で画面の中央に表示させます

text というオブジェクトでは、文字テキストのフォントサイズ、色、表示場所を指定しています

文字の表示位置も、Rect によって幾何情報を保持している点に注目しましょう

font.render()（これは font オブジェクトの render メソッド）は、指定文字を描画した Surface を返しますので、その文字を描画した Surface を、下地の Surface の上に重ねて描画する必要があります

surface.blit()（これは surface オブジェクトの blit メソッド）は、画像を他の画像の上に重ねて描画するメソッドです

ソースコード 2.11　文字テキストの表示など

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT,Rect,KEYDOWN,K_RIGHT,K_LEFT
4  import random
5
6  def fine():
7      pygame.quit()
8      sys.exit()
9
10 def key_event():
11     for event in pygame.event.get():
12         if event.type == QUIT:
13             fine()
14         elif event.type == KEYDOWN:
15             if event.key == K_RIGHT and racket.right < WIDTH:
16                 racket.centerx += 15
17             if event.key == K_LEFT and racket.left > 0:
18                 racket.centerx -= 15
19
20 pygame.init()
21 WIDTH = 640
22 HEIGHT = 480
23 WSIZE = (WIDTH, HEIGHT)
24 surface = pygame.display.set_mode( WSIZE )
25 pygame.display.set_caption( 'Squash' )
26 clock = pygame.time.Clock()
27 FPS = 20
28
29 font = pygame.font.Font(None, 60)
30 text = font.render( "Game Over!", True, (10,10,10) )
31 textpos = text.get_rect()
32 textpos.centerx = surface.get_rect().centerx
33 textpos.centery = surface.get_rect().centery
34
35 WHITE = (255,255,255)
36 RED = (255,0,0)
37 RADIUS = 10
```

```
x = random.randint( 0, WIDTH-1 )
y = 0
delx = dely = 5
RorL = random.randint(0,1)
if RorL == 0:
    delx = -delx
GREEN = (0,255,0)
racket = Rect( (WIDTH//2, HEIGHT-50, 80, 10) )
point = 0
while True:
    key_event()
    surface.fill( WHITE )
    pygame.draw.rect(surface, GREEN, racket)
    pygame.draw.circle( surface, RED, (x,y), RADIUS )
    ball_rect = Rect( (x-RADIUS, y-RADIUS, RADIUS*2, RADIUS*2) )
    if racket.colliderect( ball_rect ):
        point += 10
        dely = -dely
    if y<0:
        dely = -dely
    if x>=WIDTH or x<0:
        delx = -delx
    if y>=HEIGHT:
        surface.blit( text, textpos )
        # break
    x += delx
    y += dely
    pygame.display.set_caption( 'Squash POINT=' + str(point) )
    pygame.display.update()
    clock.tick( FPS )

fine()
```

ボールを打ち返し損じた場合、while のループを break で抜けて fine() 処理に入ることでプログラムを終了するようにしているので、せっかくのメッセージを見ることができません（break をコメントアウトして、終了の検出はマウスで Window を閉じるイベントを発生させることに頼るようにしたらよいかもしれません）

ボールを放出する位置のほか、ボールを放出する方向（左右）も乱数を使って決めています（具体的には、ボール移動の x 方向の移動量の符号を乱数で決めています）

ラケットでボールをはね返したら 10 ポイントを加算することとし、キャプションにその時点での得点を表示するようにしています

【演習】

(1) 乱数を使って、角度 30 度〜150 度（=180 度ー 30 度）の範囲でボールが放出されるように直してみましょう
(2) 青いボールを追加して放出し、2 つのボールをラケットで打ち返すゲームに直してみましょう

2.2　オブジェクト指向

ここまでに作成してきたプログラムを、オブジェクト指向の考え方を使って書き換えていきます

まず、このゲームを構成しているもの（オブジェクト：Object）を列挙して、それらのもの（オブジェクトあるいはインスタンスとも呼ばれる）が具体的にどのような特性（プロパティー：Property）を持っているのかを考えてみます

とりあえず思いつくものをあげてみると、次の 4 つでしょうか

- ボール：色、形（円）、大きさ（半径）、位置（x,y 座標）、スピードなど
- ラケット：色、位置（x,y 座標）など
- 表示するメッセージ：文字列（”Game Over!”）、色、フォント、サイズ、表示位置（x,y 座標）など
- スカッシュを行う空間（ボールが飛び交う場）：背景色、大きさ（幅、高さ）、キャプションなど

基本的に、それぞれのオブジェクトは自分自身の特性（プロパティ）を知って（保持して）いますし、自分自身のプロパティを操作するメソッド（関数）を持っています。一方、自分以外のオブジェクトのことは分かりません。つまり他のオブジェクトの情報は持たないようにして、それぞれのオブジェクトが、できるだけ独立して存在させるようにすると見通しがよくなります。

しかし実際のゲームを考えると、ボールのオブジェクトとラケットのオブジェクトの間の干渉や、スカッシュ競技場の壁にボールが当たったかどうか、またボールが後ろ（下）の壁を通り越してしまい、ゲーム終了になったかどうかなど、これらオブジェクトとオブジェクトの間の相互の関わりを調べて操作する必要が生じてきます。

そこで、ゲームの進行を司るものとして Game というオブジェクトを、上の 4 つのオブジェクトに追加して、このゲームを構成することにしましょう

オブジェクトの設計図のことをクラス（Class）といいます

ここから、上で述べた 5 つのオブジェクトそれぞれについて、クラスの設計を進めていきます

なお、この後クラスから生成される具体的なものを、オブジェクトではなく、インスタンスと呼ぶことがあります

2.2.1　main.py の処理

if __name__ == '__main__':とかくと、この下の行からプログラムの実行を開始することになります

ここでは、Game クラスのオブジェクトである game を生成し、その後 game オブジェクトの中の start() というメソッドを実行するように指示しています

ソースコード 2.12　main.py

```
1 if __name__ == '__main__':
2     game = Game()
3     game.start()
```

実際に Game クラスを記述していく際には、start() メソッドが呼び出されるのですから、Game クラスの中に start() メソッドを用意しなければなりません

仮に start() メソッドで print() 関数だけを実行させることにしますと

ソースコード 2.13　main.py と Game.py

```
1 class Game:
2     def start(self):
3         print('Gameクラスのstart()メソッド')
4
5 if __name__ == '__main__':
6     game = Game()
7     game.start()
```

クラスの中に、def __init__(self):として、予め決まった名前のメソッドを定義することができます

このメソッドはコンストラクタと呼ばれ、クラスのオブジェクトが生成される際、最初に一度だけ実行されます

start() メソッドの実行文を#でコメントアウトしてしまい、game オブジェクトの生成だけ行ってみると、コンストラクタだけが実行されることが分かります

また、start() メソッドのコメントを外してみると、コンストラクタ実行後に start() メソッドが実行されることを確認できます

ソースコード 2.14　main.py と Game.py

```
1 class Game:
2     def __init__(self):
3         print('Gameクラスのコンストラクタ')
4
5     def start(self):
6         print('Gameクラスのstart()メソッド')
7
8 if __name__ == '__main__':
9     game = Game()
10    # game.start()
```

さて、ゲームのプログラム作りに話を戻します

main プログラムを、まずは次の通り書くことにします

ソースコード 2.15　main.py

```
1 from Game import Game
2
3 if __name__ == '__main__':
4     game = Game()
5     game.start()
```

2.2.2　画面のクラス：Screen

Screen クラスでは、スカッシュを行う場所、ボールの飛び交う場を定義します

このクラスが持つ主な特性値（プロパティ）は次の通りです

- 画面の幅（WIDTH）
- 画面の高さ（HEIGHT）

- 描画面（surface）

これらの特性値はコンストラクタ（__init__() メソッド）の中で、それぞれの初期値を設定しています
コンストラクタの引数には、デフォルトの値を設定しています
これらの特性値を操作するメソッドは 2 つです

- fill メソッドは、
  fill メソッドによって、引数で受け取った color 色で surface を塗りつぶします
- caption メソッドは、
  set_caption メソッドによって、引数で受け取った文字列を Window のタイトルバーに表示します

ソースコード 2.16　Class Screen

```
1  import pygame
2
3  class Screen:
4      def __init__(self, width=600, height=600):
5          self.WIDTH = width
6          self.HEIGHT = height
7          SIZE = (width, height)
8          self.surface = pygame.display.set_mode( SIZE )
9
10     def fill(self, color=(255, 255, 255)):
11         self.surface.fill( color )
12
13     def caption(self, str):
14         pygame.display.set_caption( str )
```

2.2.3　ゲームの進行を司るクラス：Game

ゲームの進行を司るクラス Game は、今の段階では、そのコンストラクタの中で Screen クラスのオブジェクトを screen という名前で生成し、それを表示しています

Game クラスの key_event() メソッドでは、Window を閉じる時のイベント QUIT を取得すると、終了処理 fine() メソッドを呼び出しています

start() メソッドは、このゲームの主な処理（繰り返し）を行うメソッドなので、この中に具体的なゲームの進行を記述していきます

この後、この Game クラスを少しずつ改造していきます

ソースコード 2.17　Game クラス（screen オブジェクトを生成）

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4  from Screen import Screen
5
6  class Game():
7      WHITE = (255, 255, 255)
8      def __init__(self):
9          pygame.init()
10         self.WIDTH = 640
```

```
11          self.HEIGHT = 480
12          self.screen = Screen( self.WIDTH, self.HEIGHT )
13          self.screen.caption( "Squash game" )
14          self.clock = pygame.time.Clock()
15          self.FPS = 30
16
17      def fine(self):
18          pygame.quit()
19          sys.exit()
20
21      def key_event(self):
22          for event in pygame.event.get():
23              if event.type == QUIT:
24                  self.fine()
25
26      def start(self):
27          game_over = False
28          while not game_over:
29              self.key_event()
30              self.screen.fill( Game.WHITE )
31              # ↓　ここからゲームの進行を記述していく
32
33              # ↑　ゲーム進行の記述はここまで
34              pygame.display.update()
35              self.clock.tick(self.FPS)
```

クラスの中で定義する関数（メソッドと呼ばれる）の第 1 引数には、必ず「self」と指定します

self は、そのクラスから生成される「オブジェクトそれ自身」のことを指しています

「self. 変数名」と書いたとき、その変数はオブジェクト変数と呼ばれ、self すなわち、クラスから生成された具体的なオブジェクトの中に保持されている変数という意味になります。そのクラスから複数のオブジェクトが生成された場合には、それぞれのオブジェクトが個別に持っている変数を指していることになります

一方、「self.」を付けていない変数等は、その関数内でだけ通用するローカルな変数になりますので、当該オブジェクトの中の他のメソッドから参照することはできませんし、またそのクラスの外から参照することもできません

なお、メソッドを呼び出すときは、self を除いた形で引数を記述します

この他に、「クラス名. 変数名」と記述するクラス変数というものがあります。クラスから生成されるオブジェクトの、いずれにも共通に保持されている変数になります

ここの例では、WHITE というタプルの値をクラス変数として定義し、fill() メソッドへの引数に Game.WHITE という名前で指定しています

2.2.4　ボールのクラス：Ball

Ball のクラスが持つ特性値（プロパティ）として次のものを考えます

- ボールの色（COLOR）
- ボールの形（円は楕円 ellipse の一種）
- ボールの大きさ（円の半径 RADIUS）
- ボールの現在位置（Rect の特性値、left,top,width,height, および centerx,centery で）

- ボールの進む速さ（SPEED）
- ボールの進んでいる方向（dirx, diry）
- ボールが飛び回る場所（surface）

これらのプロパティは、コンストラクタ（__init__() メソッド）の中で、それぞれの初期値を設定しています

コンストラクタの引数には、デフォルトの値を引数として設定していますので、特別な変更を要しない限り、オブジェクト生成の際にその引数として値を明示的に書く必要がありません

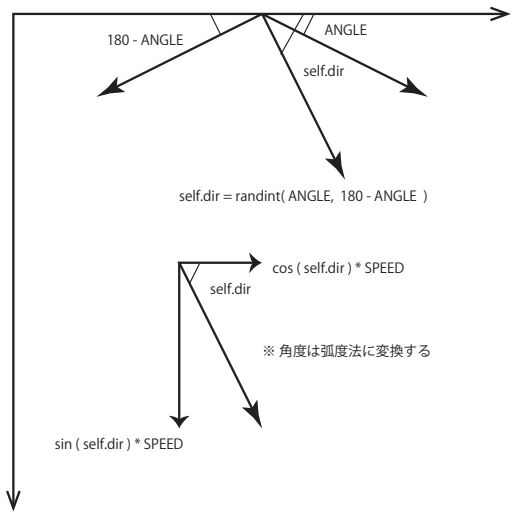
また、Ball クラスは、Rect クラスを継承しているので、Rect クラスが持っているプロパティ（left、top、centerx など）は、そのまま Ball クラスにある他のプロパティと同じように扱うことができます

これらのプロパティを操作するメソッドとして次の 5 つを用意しました

- draw メソッドは、
  ellipse 関数を使って、surface 上に Rect の特性値が保持している現在位置に、指定の色 COLOR で、円を描画します
- stop_ball メソッドは、
  ボールの SPEED をゼロにして、ゲーム終了時に呼び出されることを想定しています
- movex メソッドは、
  指定の SPEED 分だけ、現在ボールが進んでいる方向 dirx に、Rect の特性値、即ち現在位置を移動させます
- movey メソッドは、
  指定の SPEED 分だけ、現在ボールが進んでいる方向 diry に、Rect の特性値、即ち現在位置を移動させます
- movexy メソッドは、
  movex 関数と movey 関数を順に呼び出して、Rect の特性値、即ちボールの現在位置を移動させます

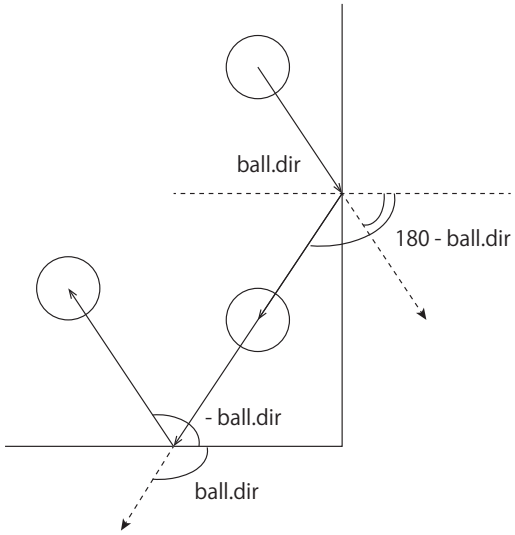
最初に放出されるボールの進行方向を、コンストラクタの中で角度 30 度〜150 度の間の乱数で作り出しています

movex() メソッドや movey() メソッドでは、ボールの進行方向の角度を、度の単位で持っていることから、それをラジアン単位に変換した上で、sin() 関数や cos() 関数を使ってボールの中心座標を計算しています



ソースコード 2.18　Ball クラス

```
1  from math import sin, cos, radians
2  from random import randint
3  import pygame
4  from pygame.locals import Rect
5
6  class Ball( Rect ):
7      def __init__(self, surface, color=(180, 180, 180),\
8                           diameter=20, speed=10, start=(300,300)):
9          self.surface = surface
10         self.COLOR = color
11         self.SPEED = speed
12         ANGLE = 30
13         self.dir = randint(ANGLE, 180 - ANGLE)
14         self.left = start[0]
15         self.top = start[1]
16         self.width = diameter
17         self.height = diameter
18
19     def stop_ball(self):
20         self.SPEED = 0
21
22     def movex(self):
23         self.centerx += int( cos(radians(self.dir)) * self.SPEED )
24
25     def movey(self):
26         self.centery += int( sin(radians(self.dir)) * self.SPEED )
27
28     def movexy(self):
29         self.movex()
30         self.movey()
31
32     def draw(self):
33         pygame.draw.ellipse(self.surface, self.COLOR, self)
```



Game クラスの書き換え

Ball クラスのオブジェクト赤い ball を、Game クラスのコンストラクタで生成し、start() メソッドの中でその ball オブジェクトを描画 ball.draw() したり、ball オブジェクトの現在位置を動かし ball.movexy() たりしています

また、Game クラスの boundary() メソッドでは、ball オブジェクトが screen オブジェクトの横幅を超えて移動しようとしたとき、また、ball オブジェクトが screen オブジェクトの縦の長さを超えて移動しようとしたときに、ball オブジェクトの進行方向を反転させています

ソースコード 2.19　Game クラス（ball オブジェクトを追加）

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT
4  from Ball import Ball
5  from Screen import Screen
6
7  class Game():
8      WHITE = (255, 255, 255)
9      def __init__(self):
10         pygame.init()
11         self.WIDTH = 640
12         self.HEIGHT = 480
13         self.screen = Screen( self.WIDTH, self.HEIGHT)
14         self.screen.caption("Squash game")
15         self.clock = pygame.time.Clock()
16         self.FPS = 30
17         RED = (255,0,0)
18         self.ball = Ball(self.screen.surface, color=RED)
19
20     def fine(self):
21         pygame.quit()
22         sys.exit()
23
24     def key_event(self):
25         for event in pygame.event.get():
26             if event.type == QUIT:
27                 self.fine()
```

```
28
29      def boundary(self, ball):
30          if not (0 < ball.centerx < self.WIDTH):
31              ball.dir = 180 - ball.dir
32          if not (0 < ball.centery < self.HEIGHT):
33              ball.dir = -ball.dir
34
35      def start(self):
36          game_over = False
37          while not game_over:
38              self.key_event()
39              self.screen.fill( Game.WHITE )
40              self.ball.draw()
41              self.boundary( self.ball )
42              self.ball.movexy()
43              pygame.display.update()
44              self.clock.tick(self.FPS)
```

2.2.5　ラケットのクラス：Racket

Racket のクラスに持たせる特性値（プロパティ）は次の通りです

- ラケットの色（COLOR）
- ラケットの形（長方形 Rect）
- ラケットの大きさ（Rect の特性値、width,height）
- ラケットの現在位置（Rect の特性値、left,top および centerx,centery で）
- ラケットを振り回す場所（surface）

これらの特性値はコンストラクタ（__init__() メソッド）で、それぞれの初期値を設定しています
コンストラクタの引数には、デフォルトの値を設定しています
Racket クラスは、Rect クラスを継承しているので、Rect クラスが持っている特性値（left,top など）は、そのまま Racket クラスの中にある特性値と同じように扱うことができます

これらの特性値を操作するメソッドとして次の 3 つを用意しました

- draw メソッドは、
  rect 関数を使って、surface 上に Rect の特性値が保持している現在位置に、指定の色 COLOR で長方形を描画します
- movex メソッドは、
  引数で受け取った x 方向の移動量 delx だけ、Rect の特性値の値、即ち現在位置を移動させます
- movey メソッドは、
  引数で受け取った y 方向の移動量 dely だけ、Rect の特性値の値、即ち現在位置を移動させます

実は、ラケットは movey() メソッドを使って動かすことは想定していないので、movey() メソッドが呼び出されることはありません

ソースコード 2.20　Racket クラス

```
1  import pygame
2  from pygame.locals import Rect
3
4  class Racket(Rect):
5      def __init__(self, surface, color=(20, 100, 150),\
6                           left=300, top=300, width=80, height=10):
7          self.surface = surface
8          self.COLOR = color
9          self.left = left
10         self.top = top
11         self.width = width
12         self.height = height
13
14     def movex(self, delx):
15         self.centerx += delx
16
17     def movey(self, dely):
18         self.centery += dely
19
20     def draw(self):
21         pygame.draw.rect(self.surface, self.COLOR, self)
```

Game クラスの書き換え

Game クラスのコンストラクタで、Racket クラスのオブジェクト racket を生成しています

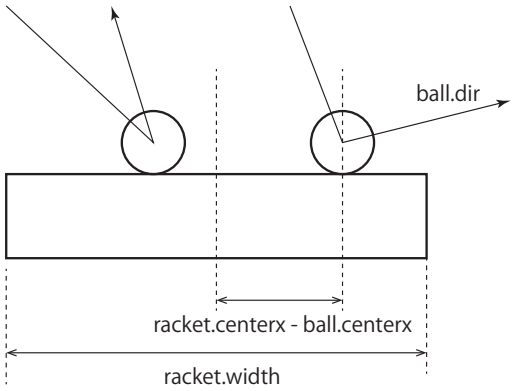
racket オブジェクトを最初に表示する位置を、left と top 変数に設定し、Racket クラスのコンストラクタの引数で渡しています

start() メソッドの中で、racket オブジェクトを描画 racket.draw() しています

key_event() メソッドでは、新たに左右の矢印キーが押下されたことを検出するイベント（K_LEFT と K_RIGHT）を捉えて、そこで、racket オブジェクトの表示位置を動かして racket.movex(移動量) います

その際、racket オブジェクトが screen オブジェクトの横幅の範囲を超えて移動させることがないように制限しています

Ball クラスも Racket クラスも、Rect クラスを継承していたので、Game クラスで定義した hitted() メソッドの中では、ball オブジェクトと racket オブジェクトが干渉したかどうかを、Rect クラスの colliderect() メソッドを使って検出することができます



ボールがラケットに当たった位置が、ラケットの中央から左右どの程度離れているかに応じて、ボールの反射する方向を変えるようにしています

ソースコード 2.21　Game クラス（racket オブジェクトを追加）

```
import sys
import pygame
from pygame.locals import QUIT, KEYDOWN, K_LEFT, K_RIGHT
from Ball import Ball
from Racket import Racket
from Screen import Screen

class Game():
    WHITE = (255, 255, 255)
    def __init__(self):
        pygame.init()
        self.WIDTH = 640
        self.HEIGHT = 480
        self.screen = Screen( self.WIDTH, self.HEIGHT)
        self.screen.caption("Squash game")
        self.clock = pygame.time.Clock()
        self.FPS = 30
        RED = (255,0,0)
        self.ball = Ball(self.screen.surface, color=RED)
        left = self.WIDTH//2
        top = self.HEIGHT - 50
        self.racket = Racket(self.screen.surface, left=left, top=top)
        pygame.key.set_repeat(10, 10)

    def fine(self):
        pygame.quit()
        sys.exit()

    def key_event(self):
        for event in pygame.event.get():
            if event.type == QUIT:
                self.fine()
            elif event.type == KEYDOWN:
                if event.key == K_LEFT and self.racket.left > 0:
                    self.racket.movex(-3)
                elif event.key == K_RIGHT and self.racket.right < self.WIDTH:
                    self.racket.movex(3)

    def hitted(self, racket, ball):
        if racket.colliderect( ball ):
            ball.dir = -(90+(racket.centerx-ball.centerx)/racket.width*100)

    def boundary(self, ball):
        if not (0 < ball.centerx < self.WIDTH):
            ball.dir = 180 - ball.dir
        if not (0 < ball.centery < self.HEIGHT):
            ball.dir = -ball.dir

    def start(self):
        game_over = False
        while not game_over:
            self.key_event()
            self.screen.fill( Game.WHITE )
```

```
54              self.ball.draw()
55              self.racket.draw()
56              self.hitted( self.racket, self.ball )
57              self.boundary( self.ball )
58              self.ball.movexy()
59              pygame.display.update()
60              self.clock.tick(self.FPS)
```

2.2.6　メッセージ表示のクラス：Message

クラスの「継承」を詳しく勉強するために、メッセージのクラスは、Message クラスと Mess クラスの 2 段構えにしてみました

Mess のクラスが持つ特性値（プロパティ）は次の通りです

- メッセージの文字列（MESSAGE）
- メッセージを表示する場所（XPOS,YPOS）
- メッセージの色（COLOR）
- メッセージを描画した画面を重ねる下地の画面（surface）
- 文字フォント（font）と、そのサイズ（SIZE）

文字フォントのサイズは、ここでは固定値（FSIZE=80）にしています
これらの特性値はコンストラクタ（__init__() メソッド）で、それぞれの初期値を設定しています
コンストラクタの引数には、デフォルトの値を設定しています
これらの特性値を操作するメソッドは 1 つだけです

- display メソッドは、
  Font の render メソッドを使って MESSAGE を描画し、それを中心座標（XPOS、YPOS）の位置に、COLOR 色で surface に重ね合わせます

Mess クラスを上位のクラスとして継承している Message クラスは、そのコンストラクタで上位のクラス super() のコンストラクタを呼び出しています Mess クラスが保持している MESSAGE プロパティ、XPOS、YPOS プロパティに値を設定しています
また、Game クラスの中から Message クラスの display() メソッドを呼び出していますが、実際は、継承している上位のクラス Mess が保持している display() メソッドが実行されます

ソースコード 2.22　Message クラス

```
1  import pygame
2
3  class Mess:
4      def __init__(self, surface, size=80, color=(255,255,0)):
5          self.surface = surface
6          self.SIZE = size
7          self.COLOR = color
8          self.font = pygame.font.Font(None, size)
9          self.MESSAGE = 'Hello'
10         self.XPOS = self.YPOS = 0
11
```

```
12      def display(self):
13          text = self.font.render(self.MESSAGE, True, self.COLOR)
14          textpos = text.get_rect()
15          textpos.centerx = self.XPOS
16          textpos.centery = self.YPOS
17          self.surface.blit(text, textpos)
18
19  class Message( Mess ):
20      def __init__(self, surface, message, xpos, ypos, size=80, color=(0,0,0)):
21          super().__init__(surface, size, color)
22          self.MESSAGE = message
23          self.XPOS = xpos
24          self.YPOS = ypos
```

Game クラスの書き換え

Message クラスの msg_gover オブジェクトを、Game クラスのコンストラクタで生成しています

その際、メッセージ'Game Over!!' の表示色を黄色 YELLOW に、表示位置を xpos と ypos で設定したオブジェクトにしています

コンストラクタでは、メッセージの各情報を設定しただけで、まだ表示していません

実際に表示するのは、boundary() メソッドの中で ball オブジェクトが racket オブジェクトに当たらずに、ball オブジェクトの centery プロパティが、screen オブジェクトの HEIGHT プロパティーを超えた時に、msg_gover オブジェクトが持っていた MESSAGE プロパティ'Game Over!!' を表示し、同時に、ball オブジェクトを静止 ball.stop_ball() させています

ソースコード 2.23　Game クラス（message オブジェクトを追加）

```
1   import sys
2   import pygame
3   from pygame.locals import QUIT, KEYDOWN, K_LEFT, K_RIGHT
4   from random import randint
5   from Ball import Ball
6   from Message import Message
7   from Racket import Racket
8   from Screen import Screen
9
10  class Game():
11      WHITE = (255, 255, 255)
12      def __init__(self):
13          pygame.init()
14          self.WIDTH = 640
15          self.HEIGHT = 480
16          self.screen = Screen( self.WIDTH, self.HEIGHT)
17          self.screen.caption("Squash game")
18          self.clock = pygame.time.Clock()
19          self.FPS = 30
20          RED = (255,0,0)
21          START = ( randint(0,self.WIDTH-1),0 )
22          self.ball = Ball( self.screen.surface, color=RED, start=START )
23          left = self.WIDTH//2
24          top = self.HEIGHT - 50
25          self.racket = Racket(self.screen.surface, left=left, top=top)
26          xpos = left
27          ypos = self.HEIGHT//2
```

```
28          YELLOW = (255,255,0)
29          self.msg_gover = Message( self.screen.surface, 'Game Over!!',\
30                                    xpos, ypos, color=YELLOW )
31          pygame.key.set_repeat(10, 10)
32
33      def fine(self):
34          pygame.quit()
35          sys.exit()
36
37      def key_event(self):
38          for event in pygame.event.get():
39              if event.type == QUIT:
40                  self.fine()
41              elif event.type == KEYDOWN:
42                  if event.key == K_LEFT and self.racket.left > 0:
43                      self.racket.movex(-3)
44                  elif event.key == K_RIGHT and self.racket.right < self.WIDTH:
45                      self.racket.movex(3)
46
47      def hitted(self, racket, ball):
48          if racket.colliderect( ball ):
49              ball.dir = -(90+(racket.centerx-ball.centerx)/racket.width*100)
50
51      def boundary(self, ball):
52          if not (0 < ball.centerx < self.WIDTH):
53              ball.dir = 180 - ball.dir
54          if ball.centery < 0:
55              ball.dir = -ball.dir
56          if self.HEIGHT < ball.centery:
57              ball.stop_ball()
58              self.msg_gover.display()
59
60      def start(self):
61          game_over = False
62          while not game_over:
63              self.key_event()
64              self.screen.fill( Game.WHITE )
65              self.ball.draw()
66              self.racket.draw()
67              self.hitted( self.racket, self.ball )
68              self.boundary( self.ball )
69              self.ball.movexy()
70              pygame.display.update()
71              self.clock.tick(self.FPS)
```

完成した、OOP 版 Squash ゲームのソースプログラム

-rw-r–r– 1 mat staff 892 2 23 20:30 Ball.py
-rw-r–r– 1 mat staff 2331 2 23 20:33 Game.py
-rw-r–r– 1 mat staff 751 2 23 20:32 Message.py
-rw-r–r– 1 mat staff 540 2 23 20:31 Racket.py
-rw-r–r– 1 mat staff 368 2 23 20:29 Screen.py
-rw-r–r– 1 mat staff 86 2 23 20:29 main.py

参考文献

[1] Brett Slatkin 著、黒川利明 訳 (オライリー・ジャパン) 「Effective Python 第 2 版」
[2] 辻真吾、小林秀幸、鈴木庸氏、細川康博 著（技術評論社）「Python3 スキルアップ教科書」
[3] 金宏和實 著 (日経 BP マーケティング)「はじめる Python!, ゼロからのゲームプログラミング」
[4] 田中賢一郎 著 (インプレス R&D)「ゲームを作りながら楽しく学べる Python プログラミング」