Games with Python

2021 年 4 月 3 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

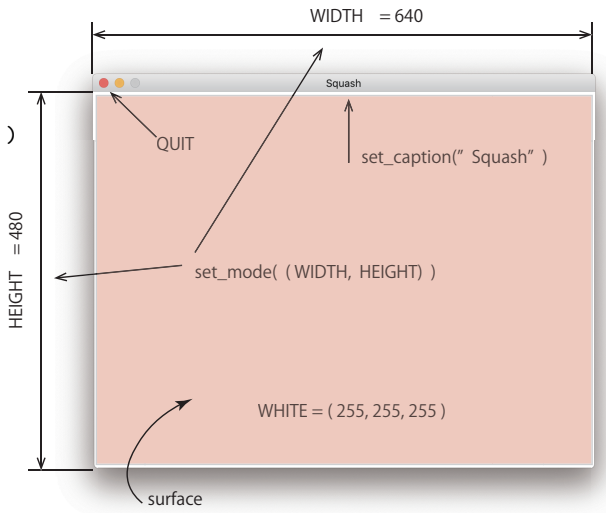
Window の surface サイズは、横幅 WIDTH と高さ HEIGHT のタプル (WIDTH, HEIGHT) で指定します。ここでは、このタプルを WSIZE という名前の変数に納めて使っています

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

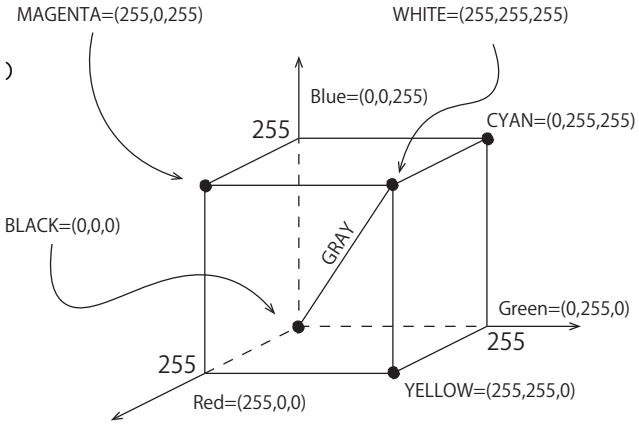
(3) キャプションを変えてみましょう

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）

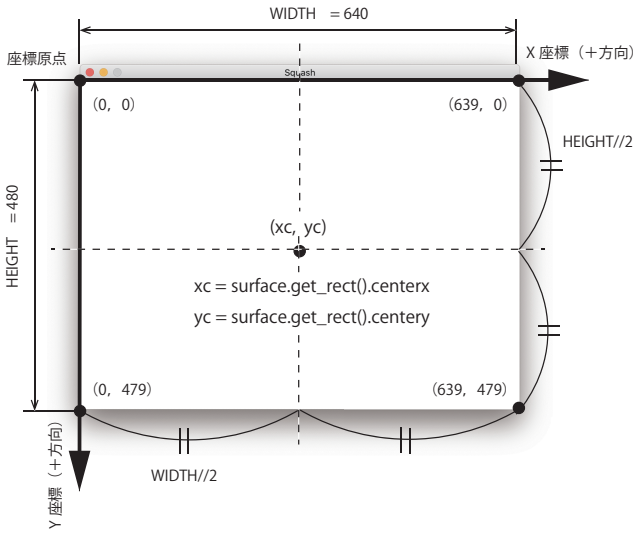
n = random.randint(0, 7) は、0≦n≦7 の範囲の整数をランダムに生成しています
確認のために、print(n)　を一行入れてみたら理解の助けになるかもしれません

【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）
- 色の措定 color は、Red,Green,Blue のタプル

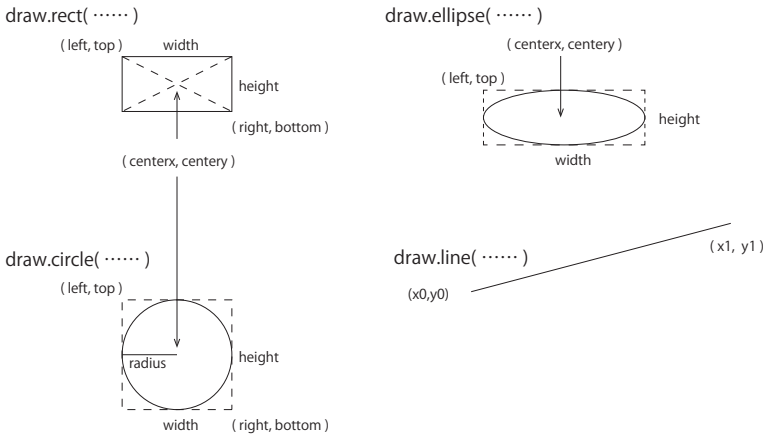
以下では、個々の図形のそのほかの引数について説明します

(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

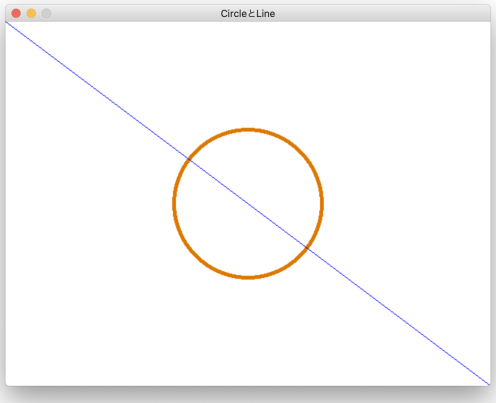


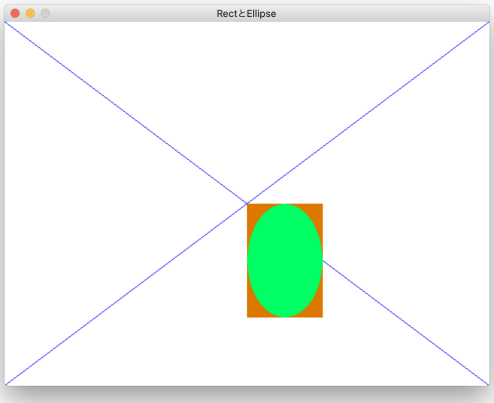
次の作図を試してみましょう

- 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

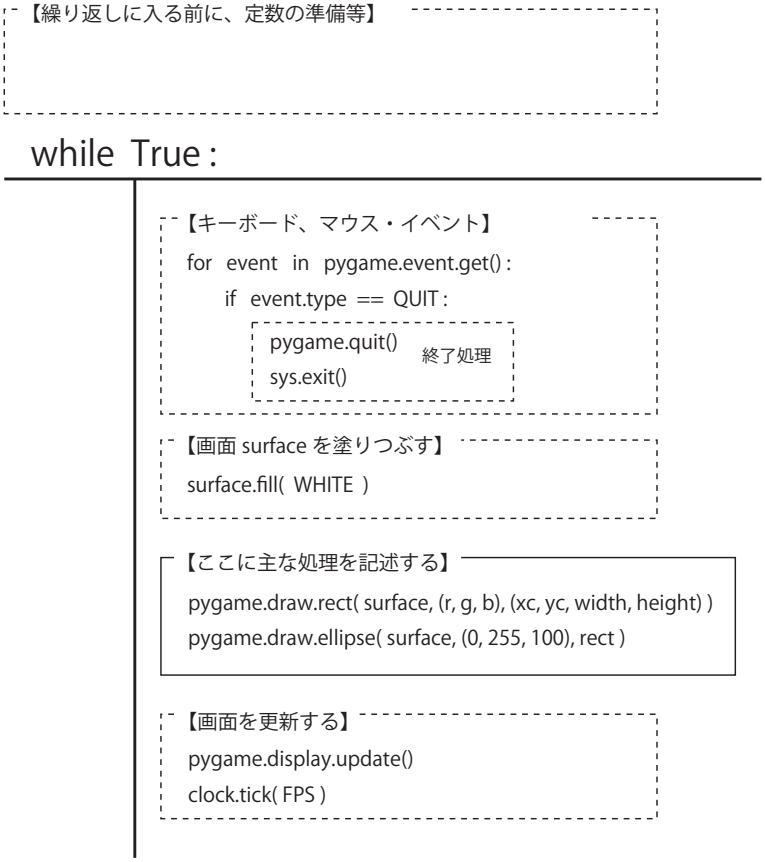
ソースコード 2.4　Rect と Ellipse

```
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しょう
(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



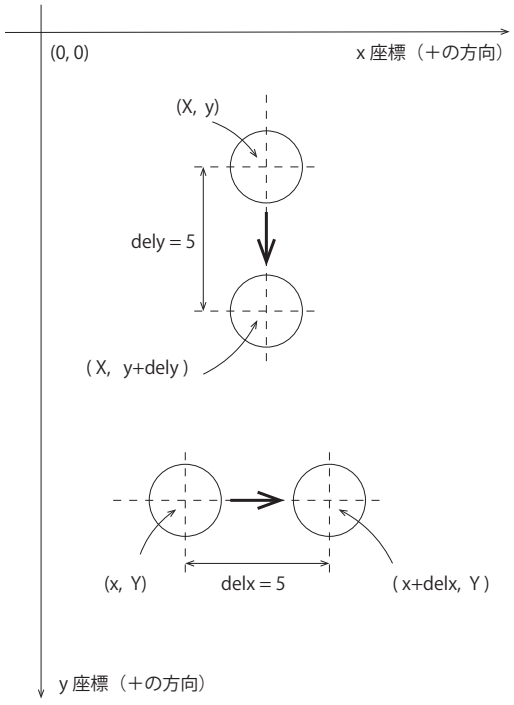
2.1.4　図形の移動

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
```

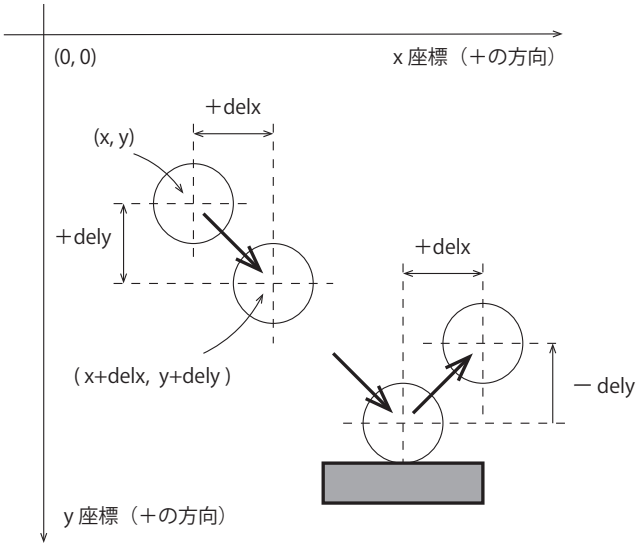
```
4
5  def fine():
6      pygame.quit()
7      sys.exit()
8
9  pygame.init()
10 WIDTH = 640
11 HEIGHT = 480
12 WSIZE = (WIDTH, HEIGHT)
13 surface = pygame.display.set_mode( WSIZE )
14 pygame.display.set_caption( 'Squash (Move Ball)' )
15 clock = pygame.time.Clock()
16 FPS = 10
17
18 WHITE = (255,255,255)
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)
```

```
RED = (255,0,0)
RADIUS = 10
x = surface.get_rect().centerx
y = 0
dely = 5

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            fine()

    surface.fill( WHITE )
    pygame.draw.circle( surface, RED, (x,y), RADIUS )
    if y>=HEIGHT or y<0:
        dely = -dely
    y += dely
    pygame.display.update()
    clock.tick( FPS )
```

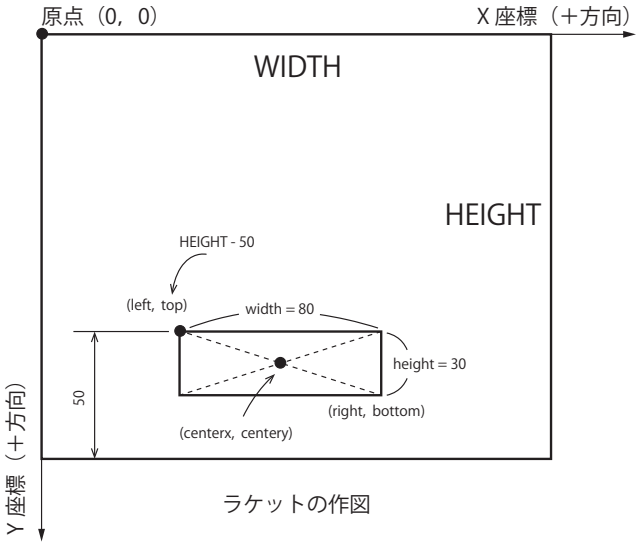
ボールが上下 y 方向で反射するように、y 方向の移動量の符号を反転しているのは前の例と同じです
ボールを左右 x 方向でも反射するように、x 方向の移動量の符号も反転させています
最初にボールを放出する位置 x 座標を、乱数で決めています

ソースコード 2.7　反射

```
import sys
import pygame
from pygame.locals import QUIT
import random

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

pygame.init()
WIDTH = 640
HEIGHT = 480
WSIZE = (WIDTH, HEIGHT)
surface = pygame.display.set_mode( WSIZE )
pygame.display.set_caption( 'Squash(Bounding Ball)' )
clock = pygame.time.Clock()
FPS = 30

WHITE = (255,255,255)
RED = (255,0,0)
RADIUS = 10
x = random.randint( 0, WIDTH-1 )
y = 0
delx = dely = 5

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            fine()

    surface.fill( WHITE )
    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
```

```
52      x += delx
53      y += dely
54      pygame.display.update()
55      clock.tick( FPS )
56
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
38 x = random.randint( 0, WIDTH-1 )
39 y = 0
40 delx = dely = 5
41 RorL = random.randint(0,1)
42 if RorL == 0:
43     delx = -delx
44 GREEN = (0,255,0)
45 racket = Rect( (WIDTH//2, HEIGHT-50, 80, 10) )
46 point = 0
47 while True:
48     key_event()
49     surface.fill( WHITE )
50     pygame.draw.rect(surface, GREEN, racket)
51     pygame.draw.circle( surface, RED, (x,y), RADIUS )
52     ball_rect = Rect( (x-RADIUS, y-RADIUS, RADIUS*2, RADIUS*2) )
53     if racket.colliderect( ball_rect ):
54         point += 10
55         dely = -dely
56     if y<0:
57         dely = -dely
58     if x>=WIDTH or x<0:
59         delx = -delx
60     if y>=HEIGHT:
61         surface.blit( text, textpos )
62         # break
63     x += delx
64     y += dely
65     pygame.display.set_caption( 'Squash POINT=' + str(point) )
66     pygame.display.update()
67     clock.tick( FPS )
68
69 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

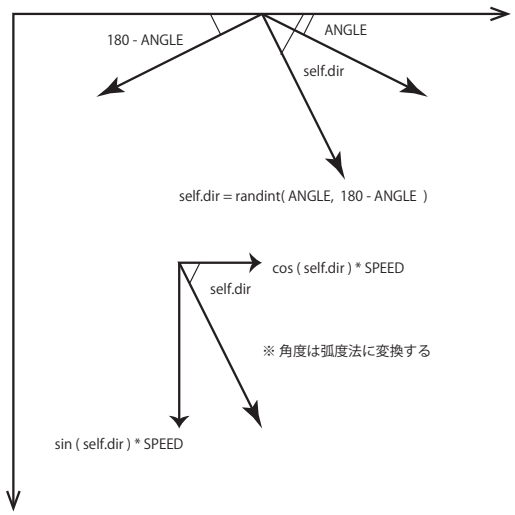
また、Ball クラスは、Rect クラスを継承しているので、Rect クラスが持っているプロパティ（left、top、centerx など）は、そのまま Ball クラスにある他のプロパティと同じように扱うことができます

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

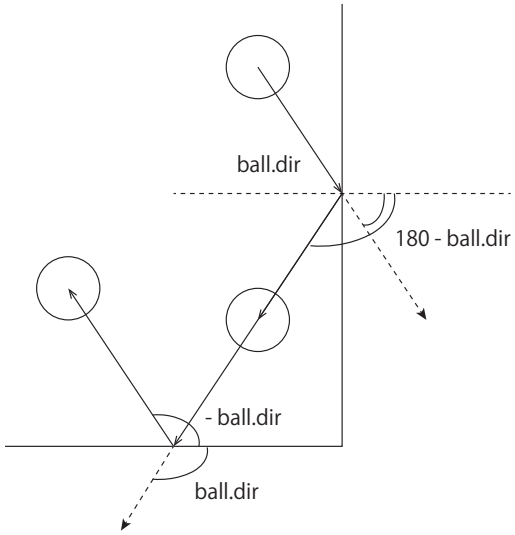
最初に放出されるボールの進行方向を、コンストラクタの中で角度 30 度〜150 度の間の乱数で作り出しています

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

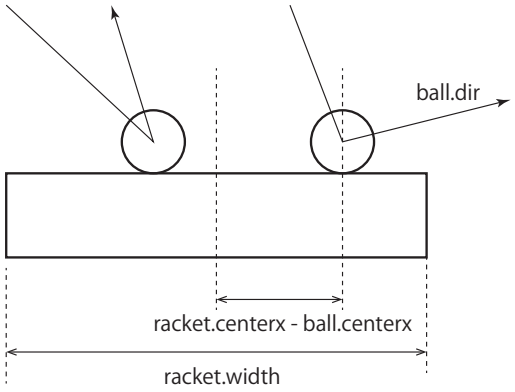
racket オブジェクトを最初に表示する位置を、left と top 変数に設定し、Racket クラスのコンストラクタの引数で渡しています

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 )
            self.ball.draw()
            self.racket.draw()
            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」を詳しく学習するために、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)
```

第 3 章

ブロック崩しゲーム

ブロック崩しゲームは、スカッシュ・ゲームに、ブロック部分のオブジェクトを追加して作ります

3.0.1　ブロッククラス：Block と Blocks

ブロック部分の全体を Blocks クラス、その中の個々のブロックを Block クラスで定義します
Block クラスは Rect クラスを継承させることとし、そのプロパティは次の通りです

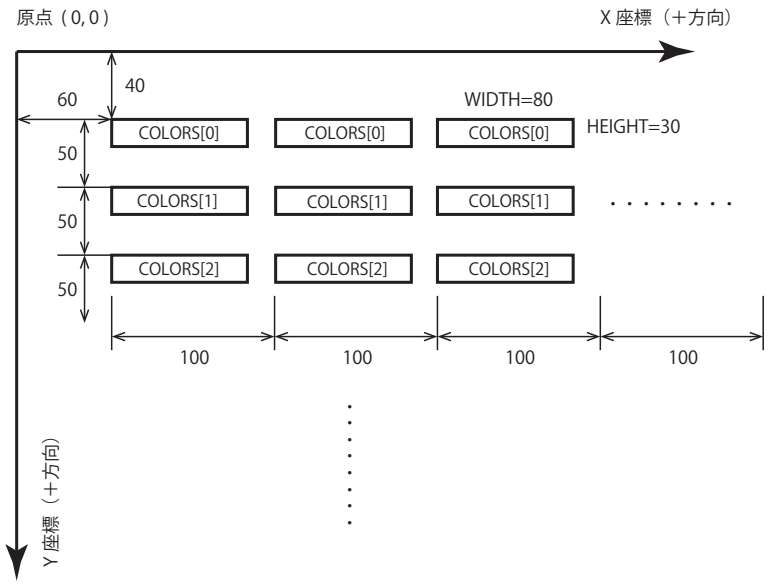
- ブロックの左座標（left）
- ブロックの上座標（top）
- ブロックの幅（width）
- ブロックの高さ（height）
- ブロックの色（COLOR）
- ブロックを配置する場所（surface）

ブロックを draw.rect() で描画するためのメソッド draw を持たせています

ソースコード 3.1　Class Block と Blocks

```
1  import pygame
2  from pygame.locals import Rect
3
4  class Block(Rect):
5      def __init__(self, surface, color, rect):
6          self.surface = surface
7          self.COLOR = color
8          self.left = rect[0]
9          self.top = rect[1]
10         self.width = rect[2]
11         self.height = rect[3]
12
13     def draw(self):
14         pygame.draw.rect(self.surface, self.COLOR, self)
15
16 class Blocks():
17     def __init__(self, surface):
18         self.surface = surface
19         WIDTH = 80
20         HEIGHT = 30
21         COLORS = [(255,0,0), (255,165,0), (242,242,0),\
22                   (0,128,0), (128,0,128), (0,0,250)]
23         self.blocks = []
24         for ypos, color in enumerate( COLORS, start=0):
25             for xpos in range(0, 5):
26                 self.blocks.append( Block(self.surface, color,\
27                             Rect(xpos*100+60, ypos*50+40, WIDTH, HEIGHT)) )
28
29     def draw(self):
30         for block in self.blocks:
31             block.draw()
```

Block オブジェクトを 6 行 5 列に並べるクラスを Blocks クラスとして定義します
Blocks クラスの主なプロパティは、Block オブジェクトのリストです



- 6 行 5 列の Block オブジェクトのリストを持っています（blocks）

メソッドとして、次のようなものを用意することが考えられます

- draw メソッド：6 行 5 列のブロックオブジェクトを描画します

オブジェクトの中のプロパティ（特性）の値を返すメソッドは「ゲッター (getter)」と呼ばれます
オブジェクトの中のプロパティ（特性）に値を設定するメソッドは「セッター (setter)」と呼ばれます
ゲッターの関数名は「get 云々」、セッターの関数名は「set 云々」という書き方をします
ゲッターやセッターを用意しようとするなら、次のメソッドを用意する必要があります

- get_blocks メソッド：6 行 5 列の Block のオブジェクトのリストを返します
- set_blocks メソッド：引数で受け取ったリストを、Blocks オブジェクト内の変数に保存します

しかし、「Effective Python」という書籍の第 6 章の項目 44 では、「get メソッドや set メソッドは使わずに属性をそのまま使う」のが良い（→その方が Pythonic な書き方である）との記述があります
Java や C++ 言語では、セッターやゲッターは通常書いてしまうものですが、Python では推奨されていない（どうしても Python で必要なら、メタクラスの書き方を使え）ということです
セッターやゲッターを使わないこととして、このクラスのメソッドは draw メソッドだけにします

Game クラスの書き換え

ブロック崩しでは、ブロックを全て打ち崩した場合、'Cleared!!' と表示させたいと思います
また、ブロックを 1 つ消すごとに 10 ポイントを加算して、それが画面上部に表示されるようにします
Blocks クラスのオブジェクトである blocks の中に保持している blocks という名称のプロパティですが、これは Block クラスのオブジェクトである block のリストでしたから、そのリストの長さを len() 関数で調べて、長さがゼロになったなら全てのブロックを崩してしまったと判断して、'Cleared!!' と表示しています

ソースコード 3.2　Game クラス（blocks 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  from Blocks import Blocks
10
11 class Game():
12     BLACK = (0, 0, 0)
13     def __init__(self):
14         pygame.init()
15         self.WIDTH = 600
16         self.HEIGHT = 800
17         self.screen = Screen( self.WIDTH, self.HEIGHT)
18         self.screen.caption("Block game")
19         self.clock = pygame.time.Clock()
20         self.FPS = 30
```

```
21          RED = (255,0,0)
22          START = ( randint(0,self.WIDTH-1), self.HEIGHT//2 )
23          self.red_ball = Ball( self.screen.surface, color=RED, start=START )
24          left = self.WIDTH//2
25          top = self.HEIGHT - 50
26          self.racket = Racket(self.screen.surface, left=left, top=top )
27          xpos = left
28          ypos = self.HEIGHT//2
29          YELLOW = (255,255,0)
30          self.msg_gover = Message( self.screen.surface, 'Game Over!!',\
31                                    xpos, ypos, color=YELLOW )
32          self.msg_clear = Message( self.screen.surface, 'Clear!!',\
33                                    xpos, ypos, color=YELLOW)
34          ypos = 20
35          self.msg_point = Message( self.screen.surface, 'Point'+'=0',\
36                                    xpos, ypos, size=30, color=YELLOW )
37          self.blocks = Blocks( self.screen.surface )
38          self.point = 0
39          pygame.key.set_repeat(10, 10)
40
41      def fine(self):
42          pygame.quit()
43          sys.exit()
44
45      def key_event(self, racket):
46          for event in pygame.event.get():
47              if event.type == QUIT:
48                  self.fine()
49              elif event.type == KEYDOWN:
50                  if event.key == K_LEFT and racket.left > 0:
51                      racket.movex(-3)
52                  elif event.key == K_RIGHT and racket.right < self.WIDTH:
53                      racket.movex(3)
54
55      def cleared(self, blocks, ball):
56          if len(blocks.blocks) == 0:
57              ball.stop_ball()
58              self.msg_clear.display()
59
60      def clashed(self, blocks, ball):
61          b0 = blocks.blocks
62          b1 = [x for x in b0 if not x.colliderect(ball)]
63          if len(b0) != len(b1):
64              blocks.blocks = b1
65              ball.dir = -ball.dir
66              self.point += 10
67              self.msg_point.MESSAGE = 'Point=' + str(self.point)
68
69      def hitted(self, racket, ball):
70          if racket.colliderect( ball ):
71              ball.dir = -(90 + (racket.centerx-ball.centerx)/racket.width*100)
72
73      def boundary(self, ball):
74          if not (0 < ball.centerx < self.WIDTH):
75              ball.dir = 180 - ball.dir
76          if ball.centery < 0:
77              ball.dir = -ball.dir
78          if self.HEIGHT < ball.centery:
79              ball.stop_ball()
```

```
80              self.msg_gover.display()
81
82      def draw(self):
83          self.screen.fill( Game.BLACK )
84          self.blocks.draw()
85          self.red_ball.draw()
86          self.racket.draw()
87
88      def start(self):
89          game_over = False
90          while not game_over:
91              self.key_event( self.racket )
92              self.draw()
93              self.cleared( self.blocks, self.red_ball )
94              self.clashed( self.blocks, self.red_ball )
95              self.hitted( self.racket, self.red_ball )
96              self.boundary( self.red_ball )
97              self.msg_point.display()
98              self.red_ball.movexy()
99              pygame.display.update()
100             self.clock.tick(self.FPS)
```

Game クラスに Screen クラスを継承させる

Game クラスが Screen クラスを継承することによって、Game クラスは Screen クラスが持っているプロパティとメソッドを、あたかも自分のクラスのもののように使うことができます

これまで、self.screen.surface と書いていた部分が何カ所かありました

これは、Screen クラスのオブジェクトである screen が保持している surface という名前のプロパティだという意味なのですが、この部分を self.surface として、あたかも Game クラスのオブジェクトが surface というプロパティを持っているかのように記述できることになります

また、surface オブジェクトの幅 WIDTH と高さ HEIGHT についても、Game の中で定義しなくても Screen クラスのプロパティを、self.WIDTH や self.HEIGHT として参照できます

Screen クラスのコンストラクタを、super().__init__(引数) という形で呼び出している点にも注目しましょう

ソースコード 3.3　Game クラス（Screen クラスを継承）

```
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  from Blocks import Blocks
10
11 class Game( Screen ):
12     BLACK = (0, 0, 0)
13     def __init__(self):
14         pygame.init()
15         super().__init__( width=600, height=800 )
16         self.caption("Block game")
```

```
17          self.clock = pygame.time.Clock()
18          self.FPS = 30
19          RED = (255,0,0)
20          START = ( randint(0,self.WIDTH-1), self.HEIGHT//2 )
21          self.red_ball = Ball( self.surface, color=RED, start=START )
22          left = self.WIDTH//2
23          top = self.HEIGHT - 50
24          self.racket = Racket(self.surface, left=left, top=top )
25          xpos = left
26          ypos = self.HEIGHT//2
27          YELLOW = (255,255,0)
28          self.msg_gover = Message( self.surface, 'Game Over!!', xpos, ypos, color=
                YELLOW )
29          self.msg_clear = Message( self.surface, 'Clear!!', xpos, ypos, color=YELLOW
                )
30          ypos = 20
31          self.msg_point = Message( self.surface, 'Point'+'=0', xpos, ypos, size=30,
                color=YELLOW )
32          self.blocks = Blocks( self.surface )
33          self.point = 0
34          pygame.key.set_repeat(10, 10)
35
36      def fine(self):
37          pygame.quit()
38          sys.exit()
39
40      def key_event(self, racket):
41          for event in pygame.event.get():
42              if event.type == QUIT:
43                  self.fine()
44              elif event.type == KEYDOWN:
45                  if event.key == K_LEFT and racket.left > 0:
46                      racket.movex(-3)
47                  elif event.key == K_RIGHT and racket.right < self.WIDTH:
48                      racket.movex(3)
49
50      def cleared(self, blocks, ball):
51          if len(blocks.blocks) == 0:
52              ball.stop_ball()
53              self.msg_clear.display()
54
55      def clashed(self, blocks, ball):
56          b0 = blocks.blocks
57          b1 = [x for x in b0 if not x.colliderect(ball)]
58          if len(b0) != len(b1):
59              blocks.blocks = b1
60              ball.dir = -ball.dir
61              self.point += 10
62              self.msg_point.MESSAGE = 'Point=' + str(self.point)
63
64      def hitted(self, racket, ball):
65          if racket.colliderect( ball ):
66              ball.dir = -(90 + (racket.centerx-ball.centerx)/racket.width*100)
67
68      def boundary(self, ball):
69          if not (0 < ball.centerx < self.WIDTH):
70              ball.dir = 180 - ball.dir
71          if ball.centery < 0:
72              ball.dir = -ball.dir
```

```
73          if self.HEIGHT < ball.centery:
74              ball.stop_ball()
75              self.msg_gover.display()
76
77      def draw(self):
78          self.fill( Game.BLACK )
79          self.blocks.draw()
80          self.red_ball.draw()
81          self.racket.draw()
82
83      def start(self):
84          game_over = False
85          while not game_over:
86              self.key_event( self.racket )
87              self.draw()
88              self.cleared( self.blocks, self.red_ball )
89              self.clashed( self.blocks, self.red_ball )
90              self.hitted( self.racket, self.red_ball )
91              self.boundary( self.red_ball )
92              self.msg_point.display()
93              self.red_ball.movexy()
94              pygame.display.update()
95              self.clock.tick(self.FPS)
```

【演習】

(1) ゲームを Clear できずに Game over となった場合、画面の背景色を変えて（例えば白黒点滅させて）終わるような演出を考えてみよう
(2) Ball クラスに、「ボールの移動スピードを変えるメソッド」を追加し、
得点（取得ポイント）が増えるのに従って、ボールのスピードが速くなるように直してみよう
(3) Ball クラスに、「ボールの半径を変えるメソッド」を追加し、
得点（取得ポイント）が増えるのに従って、ボールの半径が小さくなるように直してみよう
(4) Ball クラスに、「ボールの色を変えるメソッド」を追加し、
ラケットで跳ね返すタイミングで、乱数によってボールの色を変えるように演出してみよう
(5) Racket クラスに、「ラケットの色を変えるメソッド」を追加し、
ボールをラケットで打ち返す瞬間、ラケットの色、そしてボールの色や半径を一時的に変えて（点滅させて）元に戻すなどの演出を考えてみよう
(6) Racket クラスに、「ラケットの横幅を変えるメソッド」を追加し、
得点（取得ポイント）が増えるのに従って、ラケットの幅が短くなっていくように直してみよう
(7) 得点（取得ポイント）が増えるのに従って、ラケットを移動させる単位移動ステップ数が少なくなるように直してみよう
(8) blue_ball オブジェクトを追加して、ゆっくり移動する blue_ball と速く移動する red_ball の 2 個を、ラケットで跳ね返しながらゲームできるように直してみよう
(9) 得点（取得ポイント）がある点数以上になったら、2 個目のボールが出現するように直してみよう
(10) 2 つのボールの干渉を検出できるようにして、衝突したときの両方のボールの動作を考えてみよう
(11) 通常スピードで動く red_ball の 1 個だけで行う Stage1 を Clear できたなら、Stage2 では 2 個目のボールが追加された状態でゲームが再開されるように直してみよう
(12) 2 個のボールでゲームをスタートし、途中ブロックの数が特定の値になっている間に、2 個のボールを衝突させた場合に、2 個のボールが合体して 1 個のボールでゲームを進行できるように直してみよう（ボールが合体するときには、ボールの色や半径を一時的に変化させる演出を考えてみよう）
(13) ブロックの数が最後の 1 個になった時に、にわかに数個のボールが出現するようにする、、と、、、、Clear できないゲームになるかも
(14) 実は、pygame では音を出すこともできます
ラケットにボールが当たったときの音やブロックにボールが当たったときの音、Game over のときの音やゲームを Clear できたときの音など、さらには BGM などの演出も考えられますので、調べて試してみると楽しいゲームになるかもしれませんね

第 4 章

スペース インベーダ ゲーム

4.1　スペース インベーダ ゲーム

4.1.1　画面のクラス：Screen

ソースコード 4.1　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          self.SIZE = (width, height)
8          self.surface = pygame.display.set_mode(self.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)
```

4.1.2　ゲームの進行を司るクラス：Game

ソースコード 4.2　Class Game

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, KEYDOWN, K_RIGHT, K_LEFT, K_SPACE
4  from Screen import Screen
5
6  class Game( Screen ):
7      BLACK = (0, 0, 0)
8      def __init__(self):
9          pygame.init()
10         super().__init__(width=600, height=800)
11         self.caption("Invader game")
12         self.clock = pygame.time.Clock()
13         self.FPS = 30
14
15     def fine(self):
16         pygame.quit()
17         sys.exit()
18
19     def key_event(self):
20         for event in pygame.event.get():
21             if event.type == QUIT:
22                 self.fine()
23
24     def start(self):
25         self.gameover = False
26         while True:
27             self.key_event()
28             self.fill( Game.BLACK )
29             # ここにゲームの流れを記述します
30             pygame.display.update()
31             self.clock.tick(self.FPS)
```

Game クラスは Screen クラスを継承するようにします
クラス変数 Game.BLACK を用意しています
なお、main.py を次のようにします

ソースコード 4.3　main.py

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

4.1.3　画像データを管理するクラス：Images

Ship 以外の画像は画面に動きをつけるため、2 つの画像を交互に描画する事を想定しています

このクラスは、他のクラス（ビーム、爆弾、エイリアン、自機）の上位クラスとして使います

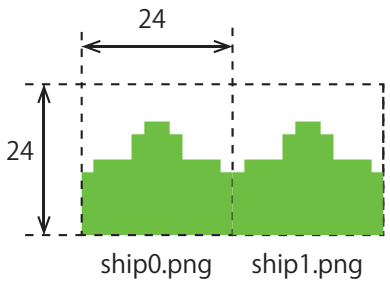
それぞれ下位クラスがコンストラクタで初期化される際に呼び出され、2 つの画像を読み込んで images[0] と images[1] に納めています

なお、画像データは全て 24 x 24 ピクセルのサイズを想定しています

ソースコード 4.4　Class Images

```
1  import pygame
2  from pygame.locals import Rect
3
4  class Images( Rect ):
5      def __init__(self, surface, fname0, fname1):
6          self.surface = surface
7          self.count = 0
8          width = height = 24
9          self.size = (width, height)
10         strip0 = pygame.image.load(fname0 + '.png')
11         strip1 = pygame.image.load(fname1 + '.png')
12         IMGS = (pygame.Surface(self.size, pygame.SRCALPHA), \
13                 pygame.Surface(self.size, pygame.SRCALPHA))
14         top = left = 0
15         rect = Rect(left, top, width, height)
16         IMGS[0].blit( strip0, (0,0), rect )
17         IMGS[1].blit( strip1, (0,0), rect )
18         self.images=[IMGS[0], IMGS[1]]
19
20     def move(self, delx, dely):
21         self.count += 1
22         self.move_ip(delx, dely)
23
24     def draw(self):
25         self.count = self.count % 2
26         image = self.images[ self.count ]
27         self.surface.blit(image, self.topleft)
```

4.1.4　自機のクラス：Ship



ship0.png と ship1.png に同じ画像データを使うと、Ship の表示は変わらないまま移動させられます

ソースコード 4.5　Class Ship

```
from Images import Images

class Ship( Images ):
    def __init__(self, surface):
        super().__init__(surface, 'ship0', 'ship1')
        self.surface = surface
        self.left = surface.get_rect().width//2
        self.top = surface.get_rect().height - 50
        self.width = self.height = 24

    def movex(self, delx):
        self.move(delx, 0)
```

movex() メソッドの引数 delx は、自機の移動速度です

Game クラスの書き換え

Game クラスのコンストラクタで、Ship クラスのオブジェクトを生成します
KEYDOWN イベントを取得し、左右方向の矢印で ship オブジェクトの位置を左右に動かします
draw メソッドによって、ship オブジェクトを描画します

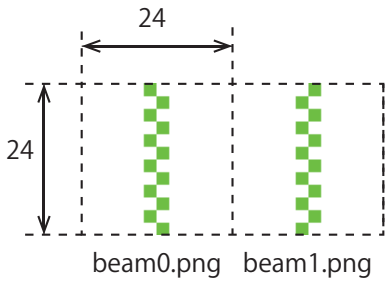
ソースコード 4.6　Game クラス（ship オブジェクトを追加）

```
import sys
import pygame
from pygame.locals import QUIT, KEYDOWN, K_RIGHT, K_LEFT, K_SPACE
from Screen import Screen
from Ship import Ship

class Game( Screen ):
    BLACK = (0, 0, 0)
    def __init__(self):
        pygame.init()
            ・
            ここは変わらず
            ・
        self.FPS = 30
        self.ship = Ship( self.surface )  # Shipクラスのオブジェクト生成を追加

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

    def key_event(self, ship):  # ship を引数に追加
        for event in pygame.event.get():
            if event.type == QUIT:
                self.fine()
            elif event.type == KEYDOWN:     # キーボードイベントの取得を追加
                if event.key == K_LEFT and ship.left > 0:
                    ship.movex(-5)     # shipオブジェクトの左右の動きを追加
                elif event.key == K_RIGHT and ship.right < self.WIDTH:
                    ship.movex(5)

    def start(self):
```

```
32          self.gameover = False
33          while True:
34              self.key_event( self.ship )    # self.ship を引数に追加
35              self.fill( Game.BLACK )
36              #
37              self.ship.draw()          # shipオブジェクトの描画を追加
38              #
39              pygame.display.update()
40              self.clock.tick(self.FPS)
```

【演習】ここまでのプログラムを実行してみよう
(自機 Ship が左右の矢印キーのイベントで移動できることを確認しよう)

4.1.5　ビームのクラス：Beam



beam0.png と beam1.png を交互に表示して、動きを表します
ビームのオブジェクトは、画面の中に 1 個だけです
movey() メソッドのデフォルト引数（dely=-15）は、ビームオブジェクトの移動速度です

ソースコード 4.7　Class Beam

```
1  from Images import Images
2
3  class Beam( Images ):
4      def __init__(self, surface):
5          super().__init__(surface, 'beam0', 'beam1')
6          self.surface = surface
7
8      def movey(self, dely=-15):
9          self.move(0, dely)
```

Game クラスの書き換え

Beam クラスのオブジェクトを、Game クラスのコンストラクタの中で生成します
キーボードイベントでスペースキーの押下（K_SPACE）を検出したとき、beam オブジェクトの下端（beam.bottom）の値が負になっている（<0）なら、beam オブジェクトは画面の外にいるので、ビームを、自機 (ship) から発射させることができます（beam.center=ship.center）
その後ゲームの反復ループ内で、beam オブジェクトを上方向（y が減る方向）に移動させ描画します
beam オブジェクトは、そのうち画面の上端より上（beam.bottom < 0）に行って、画面から消えます
画面から消えたら、再び自機から発射することが可能になります

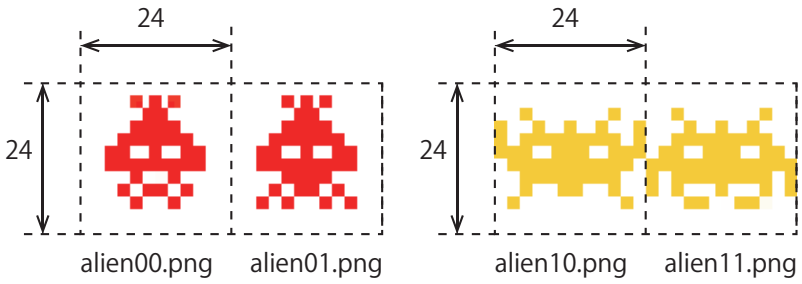
ソースコード 4.8　Game クラス（beam オブジェクトを追加）

```
import sys
import pygame
from pygame.locals import QUIT, KEYDOWN, K_RIGHT, K_LEFT, K_SPACE
from Screen import Screen
from Ship import Ship
from Beam import Beam

class Game( Screen ):
    BLACK = (0, 0, 0)
    def __init__(self):
        pygame.init()
            ・
            ここは変わらず
            ・
        self.ship = Ship( self.surface )
        self.beam = Beam( self.surface )    # beamオブジェクト生成を追加

    def fine(self):
            ・
            変わらず
            ・
    def key_event(self, ship, beam):  # beam を引数に追加
        for event in pygame.event.get():
            if event.type == QUIT:
                self.fine()
            elif event.type == KEYDOWN:
                if event.key == K_LEFT and ship.left > 0:
                        ・
                    変わらず
                        ・
                elif event.key == K_SPACE and beam.bottom < 0:
                    # beamの位置をshipの位置と一緒にして、そこからbeamを発射
                    beam.center = ship.center

    def start(self):
        self.gameover = False
        while True:
            self.key_event( self.ship, self.beam )  # self.beam を引数に追加
            self.fill( Game.BLACK )
            #
            self.ship.draw()
            self.beam.movey()    # 追加：beam を動かして
            self.beam.draw()     # 追加：beam を描画
            #
            pygame.display.update()
            self.clock.tick(self.FPS)
```

【演習】ここまでのプログラムを実行してみよう
(スペースキーの押下によって Ship の位置から beam が発せられることを確認しよう)

4.1.6　エイリアンのクラス：Alien と Aliens

alien00.png と alien01.png をエイリアン群の奥に配置し、alien10.png と alien11.png をエイリアン群の手前に配置します



エイリアンの一群を Aliens クラスで管理し、個々のエイリアンは Alien クラスで定義しています

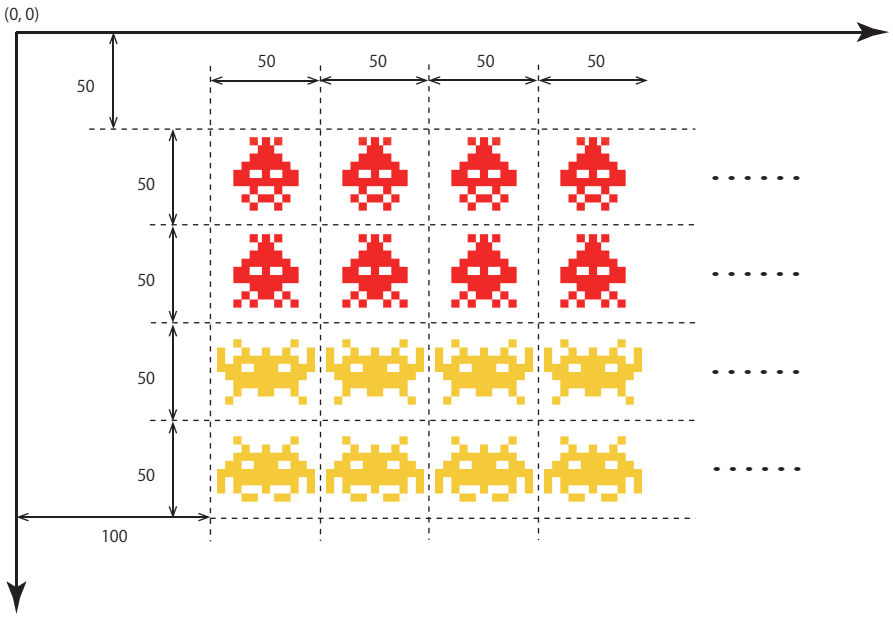
エイリアンのオブジェクトを生成するときに、alien = Alien(surface, rect, 0, (4-ypos)*10)、などとして、最後の引数でそれぞれのオブジェクト毎に得点を保持させています。

自機 Ship からスペースバーによって発射された Beam が、Alien のオブジェクトに当たったとき、その Alien が保持していた得点を獲得したポイントとして加算していくようにします。

ソースコード 4.9　Class Aliens

```
1  from Images import Images
2  from pygame.locals import Rect
3
4  class Alien(Images):
5      def __init__(self, surface, rect0, img, point):
6          if img==0:
7              super().__init__(surface, 'alien00', 'alien01')
8          else:
9              super().__init__(surface, 'alien10', 'alien11')
10         self.point = point
11         self.left = rect0.left
12         self.top = rect0.top
13         self.width = rect0.width
14         self.height = rect0.height
15
16 class Aliens(Alien):
17     def __init__(self, surface):
18         self. surface = surface
19         self.counter = 0
20         self.move_interval = 20
21         self.move_steps = -5
22         self.aliens = []
23         for ypos in range(4):
24             for xpos in range(10):
25                 left = 100 + xpos * 50
26                 top =  50 + ypos * 50
27                 width = height = 24
28                 rect = Rect(left, top, width, height)
29                 if ypos<2:
30                     alien = Alien(surface, rect, 0, (4-ypos)*10)
31                 else:
32                     alien = Alien(surface, rect, 1, (4-ypos)*10)
33                 self.aliens.append( alien )
34         self.area = self.aliens[0].copy()
35         self.stopflag = False
36
37     def movexy(self):
```

```
38          if self.stopflag:
39              return
40          self.area = self.aliens[0].copy()
41          for alien in self.aliens:
42              self.area.union_ip( alien )
43          self.counter += 1
44          if self.counter % self.move_interval == 0:
45              movey = 0
46              delx =  abs(self.move_steps)
47              if self.area.left<delx or\
48                          self.area.right>self.surface.get_rect().width-delx:
49                  movey = 24
50                  self.move_steps = -self.move_steps
51                  rate = 1.0 - (self.area.top / self.surface.get_rect().height)
52                  val = int(self.move_interval * rate)
53                  self.move_interval = 1 if val<=1 else val
54              movex = self.move_steps
55              for alien in self.aliens:
56                  alien.move(movex, movey)
57
58      def draw(self):
59          for alien in self.aliens:
60              alien.draw()
61
62      def stop(self):
63          self.stopflag = True
```



movexy() メソッドが呼ばれる度に、counter をインクリメントして、counter の値が move_interval に設定した回数に達したとき（counter を move_interval で割った余りがゼロになったとき）に、エイリアンを move_steps だけ移動させるようにしています

move_interval の値は、エイリアンの群が画面を一段階降りてくるタイミングで、エイリアン群の上端の位置の画面の高さに対する割合に応じて変わるように設定しています。エイリアンが下に降りてくれば来るほど、move_interval の値は小さくなりますが、最小値は 1 です

Game クラスの書き換え

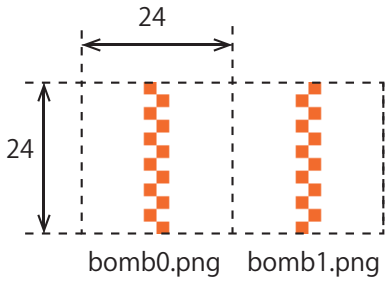
Game クラスのコンストラクタの中で、Aliens クラスのオブジェクトを生成します
ゲームの反復ループ内で、Aliens を移動させ、描画します

ソースコード 4.10　Game クラス（aliens オブジェクトを追加）

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, KEYDOWN, K_RIGHT, K_LEFT, K_SPACE
4  from Screen import Screen
5  from Ship import Ship
6  from Beam import Beam
7  from Aliens import Aliens
8
9  class Game( Screen ):
10     BLACK = (0, 0, 0)
11     def __init__(self):
12         pygame.init()
13             ・
14             変更なし
15             ・
16         self.ship = Ship( self.surface )
17         self.beam = Beam( self.surface )
18         # Aliansクラスのオブジェクト生成を追加
19         self.aliens = Aliens( self.surface )
20
21     def fine(self):
22             ・
23           変更なし
24             ・
25     def key_event(self, ship, beam):
26             ・
27           変更なし
28             ・
29     def start(self):
30         self.gameover = False
31         while True:
32             self.key_event( self.ship, self.beam )
33             self.fill( Game.BLACK )
34             #
35             self.ship.draw()
36             self.beam.movey()
37             self.beam.draw()
38             self.aliens.movexy()   # 追加：aliensオブジェクトを動かして
39             self.aliens.draw()     # 追加：aliensオブジェクトを描画
40             #
41             pygame.display.update()
42             self.clock.tick(self.FPS)
```

【演習】ここまでのプログラムを実行してみよう (エイリアンの一群が移動することを確認しましょう)

4.1.7　爆弾のクラス：Bomb と Bombs



bomb0.png と bomb1.png を交互に表示して動きを表現します

爆弾は 4 個用意しており、それぞれの爆弾は時限装置であるタイマー（time）を保持しています

movey() メソッドの引数 dely は爆弾のスピードです。また movey() が呼ばれる度に counter をインクリメントしています

爆弾の初期位置を-50 に設定（top=-50）し、時限装置（timer）を 5 から 220 の範囲の整数で設定しておきます

5 回 movey() メソッドが呼ばれると、爆弾の位置は画面の中に入ってきます（爆弾の上端 top>0）

counter の値が、爆弾の持っている時限装置の値を超えたら、爆弾を落とす役目を担うエイリアンのオブジェクトを乱数で選び、エイリアンと爆弾の位置を合わせて（bomb.center=enemy.center）爆弾を投下させています

爆弾が画面の外に出たら（top が画面の高さを超える値になったら）、再び爆弾を初期位置を-50 にセットし、時限装置（time）の現在の値に乱数の値（50〜250）を加えて、新たな時限装置としています（counter の値が時限装置の値を超えた時に爆弾が投下されます）

ソースコード 4.11　Class Bombs

```
1  from random import randint
2  from Images import Images
3
4  class Bomb( Images ):
5      def __init__(self, surface):
6          super().__init__(surface, 'bomb0', 'bomb1')
7          #super().__init__(surface, 48, 72)
8          self.top = -50
9          self.left = 0
10         self.width = self.height = 24
11         self.time = randint(5, 220)
12
13 class Bombs( Bomb ):
14     def __init__(self, surface):
15         self.surface = surface
16         self.bombs = []
17         for _ in range(4):
18             self.bombs.append( Bomb(surface) )
19         self.counter = 0
20         self.stopflag = False
21
22     def movey(self, aliens, dely=10):
23         self.counter += 1
24         for bomb in self.bombs:
```

```
            if bomb.top > 0:
                if not self.stopflag:
                    bomb.move(0, dely)
            if bomb.top > self.surface.get_rect().height:
                bomb.time += randint(50, 250)
                bomb.top = -50
            if bomb.time < self.counter and bomb.top < 0:
                enemy = aliens.aliens[randint(0, len(aliens.aliens)-1)]
                bomb.center = enemy.center

    def draw(self):
        for bomb in self.bombs:
            bomb.draw()

    def stop(self):
        self.stopflag = True
```

Game クラスの書き換え

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

ゲームの繰り返しループの中で、bombs オブジェクトを移動させ、描画しています

Bombs クラスのオブジェクトを移動させる時に、aliens オブジェクトを引数で渡しているのは、aliens の中から乱数で選んだどれかの alien を起点として、Bombs クラスのオブジェクトを動かし始める必要があるからです

ソースコード 4.12　Game クラス（bomb オブジェクトを追加）

```
import sys
import pygame
from pygame.locals import QUIT, KEYDOWN, K_RIGHT, K_LEFT, K_SPACE
from Screen import Screen
from Ship import Ship
from Beam import Beam
from Aliens import Aliens
from Bombs import Bombs

class Game( Screen ):
    BLACK = (0, 0, 0)
    def __init__(self):
        pygame.init()
            ・
            変更なし
            ・
        self.ship = Ship( self.surface )
        self.beam = Beam( self.surface )
        self.aliens = Aliens( self.surface )
        # Bombsクラスのオブジェクト生成を追加
        self.bombs = Bombs( self.surface )

    def fine(self):
        ・
        変更なし
        ・
    def key_event(self, ship, beam):
        ・
        変更なし
```

```
30          ・
31      def start(self):
32          self.gameover = False
33          while True:
34              self.key_event( self.ship, self.beam )
35                ・
36                変更なし
37                ・
38              self.aliens.movexy()
39              self.aliens.draw()
40              self.bombs.movey( self.aliens ) # 追加：bombオブジェクトを動かして
41              self.bombs.draw()               # 追加：bombオブジェクトを描画
42              #
43              pygame.display.update()
44              self.clock.tick(self.FPS)
```

【演習】ここまでのプログラムを実行してみよう (爆弾が投下されることを確認しよう)

4.1.8　メッセージのクラス：Message

ソースコード 4.13　Class Messages

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

Game クラスの書き換え

Messages クラスのオブジェクトを、3 つ生成しています

Alien と Beam の干渉を clashed メソッドでチェックし、干渉があった場合は、beam を screen の外に移し、同時に aliens を再構成しています

alien.collidepoint(beam.center) は、alien の Rect の内部に、beam.center の点座標が入っているかどうかを、bool 変数で返します

また、干渉のあった alien オブジェクトが持っている得点を point 変数に加算し、messages オブジェクトに渡して、得点を表示しています

gover メソッドでは、aliens を全て倒したか、ship に bomb が当たってゲームオーバーするか、あるいは、aliens が screen の下端まで進行してしまってゲームオーバーになるかを調べています

ゲームオーバーになったら、aliens も bombs も動きを止めるようにしています

ソースコード 4.14　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_RIGHT, K_LEFT, K_SPACE
4  from Aliens import Aliens
5  from Beam import Beam
6  from Bombs import Bombs
7  from Message import Message
8  from Screen import Screen
9  from Ship import Ship
10
11 class Game( Screen ):
12     BLACK = (0, 0, 0)
13     def __init__(self):
14         pygame.init()
15             ・
16             変更なし
17             ・
18         self.aliens = Aliens( self.surface )
19         self.bombs = Bombs( self.surface )
20         # 以下、Messageクラスのオブジェクトを3つ生成して追加
21         self.gameover = False
22         xpos = self.WIDTH//2
23         ypos = self.HEIGHT//2
24         YELLOW = (255,255,0)
25         self.msg_gover = Message( self.surface,\
26                                   'Game Over!!', xpos, ypos, color=YELLOW )
27         self.msg_clear = Message( self.surface,\
28                                   'Clear!!', xpos, ypos, color=YELLOW)
29         ypos = 20
30         self.msg_point = Message( self.surface,\
31                                   'Point'+'=0', xpos, ypos, size=30, color=YELLOW )
32         self.point = 0
33
34     def fine(self):
35         ・
36         変更なし
37         ・
38     def key_event(self, ship, beam):
39         ・
40         変更なし
41         ・
42     # ゲームオーバーを判定するメソッドを追加
43     def gover(self, aliens, ship, bombs):
44         if not aliens.aliens:      # alienがいなくなったら
45             self.msg_clear.display()
46             self.gameover = True
```

```
47          if aliens.area.bottom > self.HEIGHT-50: # alienが下端に達したら
48              self.msg_gover.display()
49              self.gameover = True
50          for bomb in bombs.bombs:
51              if bomb.colliderect(ship):     # bombがshipに当たったら
52                  self.msg_gover.display()
53                  self.gameover = True
54
55      # beamがalianに当たったかどうか判定し、aliensを再構成するメソッドを追加
56      def clashed(self, aliens, beam):
57          tmp = []
58          for alien in aliens.aliens:
59              if alien.collidepoint(beam.center):
60                  self.point += alien.point        # beamがalienに当たったら
61                  beam.top = -50
62              else:
63                  tmp.append(alien)
64          aliens.aliens = tmp
65
66      def start(self):
67          self.gameover = False
68          while True:
69              self.key_event( self.ship, self.beam )
70                  ・
71                  変更なし
72                  ・
73              self.bombs.movey( self.aliens )
74              self.bombs.draw()
75              # 追加：beamとalienの干渉チェック
76              self.clashed( self.aliens, self.beam )
77              # 追加：ゲームオーバーの判定
78              self.gover( self.aliens, self.ship, self.bombs )
79              # 追加：ゲームオーバーなら動きを止める
80              if self.gameover:
81                  self.aliens.stop()
82                  self.bombs.stop()
83              # 追加：得点表示のメッセージオブジェクト
84              self.msg_point.display('Point='+str(self.point))
85              pygame.display.update()
86              self.clock.tick(self.FPS)
```

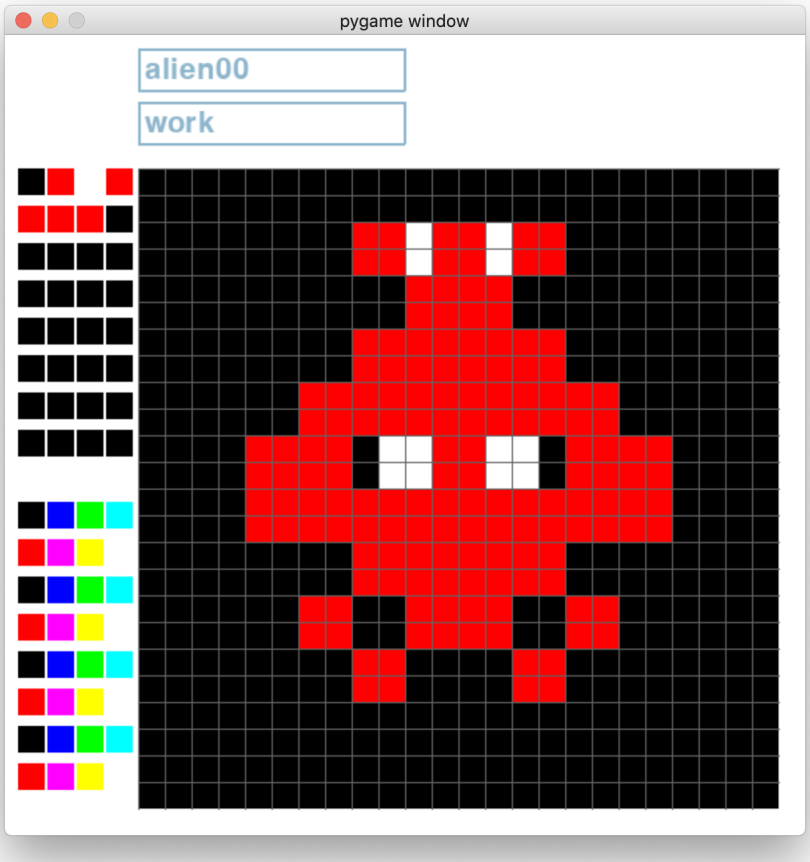
【演習】インベーダ ゲームが流行った当時のゲームでは、あるとき不意に UFO が現れ、画面の上を右から左へ、或いは左から右へと移動していきました。新たに UFO のクラスを追加し、もし UFO を打ち落としたら大きなボーナス特点が得られるようにしましょう。

【演習】UFO が爆弾を投下するようにしてみましょう

【演習】UFO がランダムに画面の途中で行き来するような動きにしてみましょう

4.2　画像データの作成、編集

インベーダゲームで使われる画像データを、自分でデザインできるようにします
24 x 24 ピクセルの画像データを想定し、画像データの編集には、Pillow ライブラリを使います



4.2.1　画面管理のクラス：Screen

作業画面の大きさと背景色を指定しています
キャプションを指定できるメソッド、画面を塗りつぶすメソッドを持っています

ソースコード 4.15　class Screen

```
1  import pygame
2
3  class Screen:
4      def __init__(self, size=(600,600), bgcolor=(0,0,0)):
5          self.surface = pygame.display.set_mode(size)
6          self.COLOR = bgcolor
7
8      def fill(self):
9          self.surface.fill(self.COLOR)
10
```

```
11      def caption(self, str):
12          pygame.display.set_caption(str)
```

4.2.2　処理の流れを司るクラス：Editor

main.py は次のようにします

ソースコード 4.16　main.py

```
1  from Editor import Editor
2
3  if __name__ == '__main__':
4      ifname = 'invader200'
5      ofname = 'work'
6      edit = Editor(ifname, ofname)
7      edit.start()
```

Editor クラスは、処理の主な流れを記述します

text0='alien00' は、編集したい入力画像のファイル名（alien00.png）を指定しています

text1='work' は、編集後の画像を保存するファイル名（work.png）を指定しています

Screen オブジェクトには、パレットが 2 つ、画像編集ボードが 1 つ、元データファイル名と保存ファイル名のそれぞれの文字入力領域を配置しています

ソースコード 4.17　class Editor

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, MOUSEBUTTONUP
4  from Board import Board
5  from Brush import Brush
6  from InputBox import InputBox
7  from Message import Message
8  from Palletes import Palletes
9  from Screen import Screen
10
11 class Editor:
12     def __init__(self, ifname='alien00', ofname='work'):
13         pygame.init()
14         self.screen = Screen(bgcolor=(255,255,255))
15         self.NUM = 24
16         xoffset = 100
17         yoffset = 100
18         size = (600, 600)
19         self.brush = Brush()
20         self.board = Board(self.screen.surface, self.brush, xoffset, yoffset, self.
               NUM, (size[0], size[1]))
21         self.text0 = text0 = ifname
22         self.text1 = text1 = ofname
23         self.board.pixels.load_image(text0)
24         self.pallete1 = Palletes(self.screen.surface, self.brush, yoff=100, type=1)
25         self.pallete2 = Palletes(self.screen.surface, self.brush, yoff=350, type=2)
26         input_box1 = InputBox(100, 10, 140, 32, text0)
27         input_box2 = InputBox(100, 50, 140, 32, text1)
28         self.input_boxes = [input_box1, input_box2]
```

```
29          self.message = Message()
30          self.clock = pygame.time.Clock()
31          self.FPS = 10
32
33      def fine(self):
34          if not self.input_boxes[1].text:
35              text1 = self.text1
36          else:
37              text1 = self.input_boxes[1].text
38          self.board.pixels.save_image('./'+text1+'.png')
39          pygame.quit()
40          sys.exit()
41
42      def key_event(self):
43          for event in pygame.event.get():
44              if event.type == QUIT:
45                  self.fine()
46              elif event.type == MOUSEBUTTONUP:
47                  if not self.board.paint(event.pos):
48                      if not self.pallete1.select(event.pos):
49                          self.pallete2.select(event.pos)
50              else:
51                  for box in self.input_boxes:
52                      box.handle_event(event)
53
54      def start(self):
55          while True:
56              self.key_event()
57              self.screen.fill()
58              self.board.draw()
59              self.pallete1.draw()
60              self.pallete2.draw()
61              for box in self.input_boxes:
62                  box.update()
63                  box.draw(self.screen.surface)
64              text0 = self.input_boxes[0].text
65              if text0!=self.text0:
66                  self.text0=text0
67                  self.board.pixels.load_image(text0)
68                  self.pallete1 = Palletes(self.screen.surface, self.brush, yoff=100,
                        type=1)
69              cstr = '(r,g,b) = ('+str(self.brush.color[0])+', '+str(self.brush.color
                    [1])+', '+str(self.brush.color[2])+')'
70              self.message.display(self.screen.surface, cstr, (320, 55))
71              pygame.display.update()
72              self.clock.tick(self.FPS)
```

4.2.3　筆のクラス：Brush

ブラシの現在色は、self.color に保持させます

パレットは、self.colors1 と self.colors2 の 2 つを持っており、それぞれに 32 色を持たせられます

self.colors2 には、自分の好きな色を登録しておけば画像編集の際にその色で描画することができます

self.colors1 には、読み込んだ画像が持っている色が登録されるようにしています

ソースコード 4.18　class Brush

```
class Brush:
    def __init__(self):
        self.color = (0,0,0)
        self.colors1 = []
        self.colors2 = []
        for _ in range(32):
            self.colors1.append((0,0,0))
            self.colors2.append((0,0,0))
        self.colors2[1] = (0,0,255)
        self.colors2[2] = (0,255,0)
        self.colors2[3] = (255,0,0)
        self.colors2[4] = (0,255,255)
        self.colors2[5] = (255,255,0)
        self.colors2[6] = (255,0,255)
        self.colors2[7] = (255,255,255)
        self.colors2[8] = (255,165,0)
        self.colors2[9] = (242,242,0)
        self.colors2[10] = (0,128,0)
        self.colors2[11] = (128,0,128)
        self.colors2[12] = (0,0,250)
```

4.2.4　画像データ描画編集用盤面クラス：Board

Board クラスは、24 x 24 ピクセルの画像編集領域です

パレットで色を選んで、現在色を brush オブジェクトの color プロパティに設定し、そのブラシで、board オブジェクトのピクセルを塗るようにします

ソースコード 4.19　class Board

```
import pygame
from pygame.locals import Rect
from Pixels import Pixels

class Board( Rect ):
    def __init__(self, surface, brush, xoff, yoff, num, size=(600, 600)):
        self.surface = surface
        self.brush = brush
        self.width = size[0]-xoff
        self.height = size[1]-yoff
        self.top = yoff
        self.left = xoff
        self.NUM = num
        self.SLOTW = self.width//num
        self.SLOTH = self.height//num
        self.fname = 'work'
        self.pixels = Pixels(self, self.fname)

    def draw(self):
        self.pixels.draw()
        GRAY = (100, 100, 100)
        for i in range(self.NUM):
            yy = self.SLOTH * i + self.top
            startp = (self.left, yy)
```

```
25              endp = (self.right-self.SLOTW, yy)
26              pygame.draw.line(self.surface, GRAY, startp, endp)
27              xx = self.SLOTW * i + self.left
28              startp = (xx, self.top)
29              endp = (xx, self.bottom - self.SLOTH)
30              pygame.draw.line(self.surface, GRAY, startp, endp)
31
32      def paint(self, pos=(-1,-1)):
33          def get_slot(xy):
34              x, y = xy[0], xy[1]
35              for ypos in range(self.NUM):
36                  y0 = self.top + self.SLOTH * ypos
37                  y1 = self.top + self.SLOTH * (ypos + 1)
38                  for xpos in range(self.NUM):
39                      x0 = self.left + self.SLOTW * xpos
40                      x1 = self.left + self.SLOTW * (xpos + 1)
41                      if y0 < y < y1 and x0 < x < x1:
42                          return xpos, ypos, ypos * self.NUM + xpos
43              return -1,-1,-1
44
45          x, y, n = get_slot(pos)
46          if 0 <= n < len(self.pixels.pixels):
47              self.pixels.pixels[n].set_color(self.brush.color)
48              self.pixels.img.putpixel((x, y), self.brush.color)
49              return True
50          return False
```

4.2.5　画素管理のクラス：Pixels、Pixel

24 x 24 ピクセルの画像の画素を管理しています

また、パレットの色の管理にも使っています

元画像データを読み込んだときに、そこで使われている色を、board オブジェクトが持っている brush オブジェクトの colors1 プロパティに設定しています

元画像の読み込みや編集後の画像の保存のためのメソッドを持っています

画像を保存する際に、黒（r=g=b=0）の画素の alpha チャネルは 0（透過）に設定しています

ソースコード 4.20　class Pixels

```
1   import pygame
2   from PIL import Image
3   from pygame.locals import Rect
4
5   class Pixel( Rect ):
6       def __init__(self, surface, color, rect):
7           self.surface = surface
8           self.COLOR = color
9           self.left = rect[0]
10          self.top = rect[1]
11          self.width = rect[2]
12          self.height = rect[3]
13
14      def set_color(self, color):
15          self.COLOR = color
16
```

```
17      def draw_pixel(self):
18          pygame.draw.rect(self.surface, self.COLOR, self)
19
20  class Pixels:
21      def __init__(self, board, fname='work'):
22          self.size=(board.NUM, board.NUM)
23          width = board.SLOTW
24          height = board.SLOTH
25          r = g = b = 255
26          self.pixels = []
27          for y in range(board.NUM):
28              top = board.SLOTH * y + board.top
29              for x in range(board.NUM):
30                  left = board.SLOTW * x + board.left
31                  px = Pixel(board.surface, (r,g,b,255), (left, top, width, height))
32                  self.pixels.append( px )
33          self.board = board
34          self.img = self.load_image(fname)
35
36      def load_image(self, fname):
37          # 元画像の読み込み
38          try:
39              im = Image.open("./"+fname+".png")
40              if im.size!=self.size:
41                  img = im.resize(self.size)
42              else:
43                  img = im
44          except Exception as f:
45              print("Error with icon file:", f)
46              img = Image.new('RGBA', self.size)
47          self.img = img
48          self.colors=[]
49          for y in range(self.img.size[1]):
50              for x in range(self.img.size[0]):
51                  col = self.img.getpixel((x, y))
52                  if col in self.colors:
53                      pass
54                  else:
55                      self.colors.append(col)
56                  pix = self.pixels[ y*self.board.NUM+x ]
57                  pix.set_color( col )
58          for n, col in enumerate(self.colors):
59              if n>len(self.board.brush.colors1)-1:
60                  break
61              else:
62                  self.board.brush.colors1[n] = col
63
64      def save_image(self, fname):
65          img2 = Image.new('RGBA', self.size)
66          for y in range(self.img.size[1]):
67              for x in range(self.img.size[0]):
68                  pix = self.pixels[y * self.img.size[1] + x]
69                  r,g,b= pix.COLOR[0], pix.COLOR[1], pix.COLOR[2]
70                  #if r==g==b==0:
71                  if r<101 and g<101 and b<101:
72                      r=g=b=a=0
73                      #a=0
74                  else:
75                      a=255
```

```
76                  img2.putpixel((x,y), (r,g,b,a))
77          img2.save(fname)
78
79      def draw(self):
80          for p in self.pixels:
81              p.draw_pixel()
```

4.2.6　パレットの管理クラス：Palletes、Pallete

ソースコード 4.21　class Palletes

```
1   import pygame
2   from pygame.locals import Rect
3
4   class Pallete( Rect ):
5       def __init__(self, surface, color, rect):
6           self.surface = surface
7           self.COLOR = color
8           self.left = rect[0]
9           self.top = rect[1]
10          self.width = rect[2]
11          self.height = rect[3]
12
13      def set_color(self, color):
14          self.COLOR = color
15
16      def draw_pallete(self):
17          pygame.draw.rect(self.surface, self.COLOR, (self.left, self.top, self.width
                , self.height))
18
19  class Palletes:
20      def __init__(self, surface, brush, xoff=10, yoff=100, numx=4, numy=8, size=(90,
             230), type=1):
21          self.surface = surface
22          self.brush = brush
23          self.width = size[0]
24          self.height = size[1]
25          self.top = yoff
26          self.left = xoff
27          self.NUMX = numx
28          self.NUMY = numy
29          self.SLOTW = self.width//numx
30          self.SLOTH = self.height//numy
31          self.palletes = []
32          self.palletegen(type)
33
34      def palletegen(self, type=1):
35          width = height = 20
36          for y in range(self.NUMY):
37              top = self.SLOTH * y + self.top
38              for x in range(self.NUMX):
39                  left = self.SLOTW * x + self.left
40                  if type==1:
41                      cl = self.brush.colors1[y*self.NUMX+x]
42                  else:
43                      cl = self.brush.colors2[y*self.NUMX+x]
```

```
44                  pallet = Pallete(self.surface, cl, (left, top, width, height))
45                  self.palletes.append(pallet)
46
47      def draw(self):
48          for p in self.palletes:
49              p.draw_pallete()
50
51      def select(self, pos=(-1,-1)):
52          def get_slot(xy):
53              x, y = xy[0], xy[1]
54              for ypos in range(self.NUMY):
55                  y0 = self.top + self.SLOTH * ypos
56                  y1 = self.top + self.SLOTH * (ypos + 1)
57                  for xpos in range(self.NUMX):
58                      x0 = self.left + self.SLOTW * xpos
59                      x1 = self.left + self.SLOTW * (xpos + 1)
60                      if y0 < y < y1 and x0 < x < x1:
61                          return ypos * self.NUMX + xpos
62              return -1
63          n = get_slot(pos)
64          if 0 <= n < self.NUMX*self.NUMY:
65              self.brush.color = self.palletes[n].COLOR
66              return True
67          return False
```

4.2.7　文字列入力のクラス：InputBox

ソースコード 4.22　class InputBox

```
1   import pygame
2
3   class InputBox:
4       def __init__(self, x, y, w, h, text=''):
5           self.COLOR_INACTIVE = pygame.Color('lightskyblue3')
6           self.COLOR_ACTIVE = pygame.Color('dodgerblue2')
7           self.FONT = pygame.font.Font(None, 32)
8           self.rect = pygame.Rect(x, y, w, h)
9           self.color = self.COLOR_INACTIVE
10          self.text = text
11          self.txt_surface = self.FONT.render(text, True, self.color)
12          self.active = False
13
14      def handle_event(self, event):
15          if event.type == pygame.MOUSEBUTTONDOWN:
16              # If the user clicked on the input_box rect.
17              if self.rect.collidepoint(event.pos):
18                  # Toggle the active variable.
19                  self.active = not self.active
20              else:
21                  self.active = False
22              # Change the current color of the input box.
23              self.color = self.COLOR_ACTIVE if self.active else self.COLOR_INACTIVE
24          if event.type == pygame.KEYDOWN:
25              if self.active:
26                  if event.key == pygame.K_RETURN:
27                      print(self.text)
```

```
28                      self.text = ''
29                  elif event.key == pygame.K_BACKSPACE:
30                      self.text = self.text[:-1]
31                  else:
32                      self.text += event.unicode
33                  # Re-render the text.
34                  self.txt_surface = self.FONT.render(self.text, True, self.color)
35
36      def update(self):
37          # Resize the box if the text is too long.
38          width = max(200, self.txt_surface.get_width()+10)
39          self.rect.w = width
40
41      def draw(self, screen):
42          # Blit the text.
43          screen.blit(self.txt_surface, (self.rect.x+5, self.rect.y+5))
44          # Blit the rect.
45          pygame.draw.rect(screen, self.color, self.rect, 2)
```

4.2.8　文字列表示のクラス：Message

ソースコード 4.23　class Message

```
1  import pygame
2
3  class Message:
4      def __init__(self, size=32, color=pygame.Color('dodgerblue2')):
5          self.font = pygame.font.Font(None, size)
6          self.color = color
7
8      def display(self, surface, mess, locate):
9          m = self.font.render(mess, True, self.color)
10         surface.blit(m, locate)
```

第 5 章

数独

ここでのプログラムは短い簡易なものなので、オブジェクト指向で記述していません
再帰呼び出しについて学習する一つの例として、数独を解くプログラムを考えます
まず、board リストに盤面を定義し、盤面の印刷を行う関数 print_board() を定義します
print_board() をそのまま呼び出して、動作を確認してみましょう

ソースコード 5.1　board

```
board = [[5,3,0,0,7,0,0,0,0],\
         [6,0,0,1,9,5,0,0,0],\
         [0,9,8,0,0,0,0,6,0],\
         [8,0,0,0,6,0,0,0,3],\
         [4,0,0,8,0,3,0,0,1],\
         [7,0,0,0,2,0,0,0,6],\
         [0,6,0,0,0,0,2,8,0],\
         [0,0,0,4,1,9,0,0,5],\
         [0,0,0,0,8,0,0,7,9]]

def print_board():
    global board
    for y in range(9):
        for x in range(9):
            print(' ',end='')
            if x in [2,5]:
                print(board[y][x], end=' |')
            else:
                print(board[y][x], end=' ')
        if y in [2,5]:
            print('\n---------|---------|--------')
        else:
            print()

# 動作確認
print_board()
```

```
 5  3  0 | 0  7  0 | 0  0  0
 6  0  0 | 1  9  5 | 0  0  0
 0  9  8 | 0  0  0 | 0  6  0
---------|---------|--------
 8  0  0 | 0  6  0 | 0  0  3
```

```
 4  0  0 | 8  0  3 | 0  0  1
 7  0  0 | 0  2  0 | 0  0  6
---------|---------|--------
 0  6  0 | 0  0  0 | 2  8  0
 0  0  0 | 4  1  9 | 0  0  5
 0  0  0 | 0  8  0 | 0  7  9
```

次に、引数 y と x で指定されたスロットに、第三引数で受け取った n を置く事ができるか否かを判定する関数 possible() を定義します

(1)　縦一列の中に、n と同じ数字があったら置けませんので、False を返します

(2)　横一行の中に、n と同じ数字があったら置けませんので、False を返します

(3)　3×3 の枠の中に、n と同じ数字があったら置けませんので、False を返します

上記（1)、(2)、(3）の何れでもないなら、True を返します

5 行 5 列目 (board リストの 0 行目や 0 列目から数え始めるので、引数は x=y=4) の空きスロットに値を指定して、possible() の動作を確認しましょう

ソースコード 5.2　possible

```
1  def possible(y,x,n):
2      global board
3      for i in range(0,9):
4          if board[y][i] == n:
5              return False
6      for i in range(0,9):
7          if board[i][x] == n:
8              return False
9      x0 = (x//3)*3
10     y0 = (y//3)*3
11     for i in range(0,3):
12         for j in range(0,3):
13             if board[y0+i][x0+j] == n:
14                 return False
15     return True
16
17 # 動作確認
18 print( possible(4,4,4) )
19 print( possible(4,4,5) )
```

最後に、問題を解く関数 solve() を定義します

y 行 x 列目のスロットに着目して、そこが 0 ならば空きスロットですから、そのスロットに、1 から 10 までの数字を順に指定して、possible() を呼びだしてはチェックしていきます

もし、possible() 関数が True を返したら、それは一つの候補ですので、引き続き solve() を繰り返して空きスロットを埋めていきます

solve() が return で None を返したときは、選ばれた候補は使えなかったという事ですので、盤面のスロットを空（0）に戻しています

全ての空欄が埋まったなら、print_board() を呼んで結果（答え）を印刷しています

ソースコード 5.3　solve

```
def solve():
    global board
    for y in range(9):
        for x in range(9):
            if board[y][x] == 0:
                for n in range(1,10):
                    if possible(y,x,n):
                        board[y][x] = n
                        solve()
                        board[y][x] = 0
                return
    print_board()

# 動作確認
solve()
```

solve() 関数の中から、自分自身である solve() 関数を呼び出す方法を再帰呼び出しと言います。再帰呼び出しを使うことによって、プログラムを短く記述できています

実行結果

```
 5  3  4 | 6  7  8 | 9  1  2
 6  7  2 | 1  9  5 | 3  4  8
 1  9  8 | 3  4  2 | 5  6  7
---------|---------|--------
 8  5  9 | 7  6  1 | 4  2  3
 4  2  6 | 8  5  3 | 7  9  1
 7  1  3 | 9  2  4 | 8  5  6
---------|---------|--------
 9  6  1 | 5  3  7 | 2  8  4
 2  8  7 | 4  1  9 | 6  3  5
 3  4  5 | 2  8  6 | 1  7  9
```

プログラムで解けてしまうとなると、しかも「こんなに短いプログラムで」解いたとなると、数独そのものはつまらないものになりますね。また「1 から 9 まで総当たりで試しながら」＝「コンピュータが力ずくで」なので、芸がないという気がします

このゲームは、まずは鉛筆と消しゴムを使って、自分で解いてみることが楽しむための重要なポイントのようです

【演習】

第 6 章

三目並べ（Tic-Tac-Toe）

三目並べは「二人零和有限確定完全情報ゲーム」と呼ばれるゲームに分類されます．簡単に言うと「勝敗は偶然に左右されず，最善手を打てばどちらかが勝つまたは引き分けになるゲーム」のことです．他にもリバーシ，チェッカー，チェス，将棋に囲碁などもこのゲームに分類されます．

初めに三目並べというゲームについて定義をしておきます．

三目並べ: 3 × 3 の格子上に二人のプレイヤーが「O」「X」を配置し、自分のマークを先に 3 つ並べた方が勝ちです．

6.1　CUI(Character User Interface) 版

6.1.1　太郎さんと花子さんの対戦

プログラムを実行すると盤面が表示され、×の石を置く場所を指定するよう促されます

画面上に示された番号を入力すると、その番号のスロットに×の石が置かれた盤面が表示され、次の手番の○に、石を置く場所を指定するように促されます

手番を交互に変えながらゲームは進み、縦、横、斜めの何れかに、先に一列に自分の石を並べた方が勝ちとなります

既に石の置かれているスロット番号を指定できませんし、スロット番号として 0 〜 8 以外の数値を指定することもできません

```
スタート！ [Tic Tac Toe]
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | 4 | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
太郎 さんの turn です。スロット番号を指定して下さい ： 4
/---|---|---\
| 0 | 1 | 2 |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
```

```
花子 さんの turn です。スロット番号を指定して下さい ： 2
/---|---|---\
| 0 | 1 | 0 |
|---|---|---|
| 3 | X | 5 |
|---|---|---|
| 6 | 7 | 8 |
\---|---|---/
太郎 さんの turn です。スロット番号を指定して下さい ：
```

ゲームの進行を司るクラス：Game

Game クラスのオブジェクトである game を生成し、game オブジェクトの中の start() というメソッドを実行します

この後、main.py と Game.py でファイルを別にしていきますので、以下のように、main.py を書いて、その冒頭で、Game.py から Game というクラスを import するという記述が必要になります

ソースコード 6.1　main.py

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

盤面を管理するクラス：Board

ゲームの盤面を保持するクラス Board を作っていきます

Board クラスのオブジェクト board を、Game クラスのコンストラクタで生成することにします

ゲームの進行を行う start() メソッドでは、board オブジェクトを印刷するメソッド draw() を呼び出すようにしています

Game クラスで、Board クラスを使っていますから、そのファイルの冒頭で、Board ファイルから Board クラスを import せよという指示が必要です

ソースコード 6.2　class Game

```
1 from Board import Board
2
3 class Game:
4     def __init__(self):
5         self.board = Board()
6
7     def start(self):
8         self.board.draw()
```

盤面の実体は、各スロットに対応させた 9 個の整数からなるリスト board で保持することとし、Board クラスのコンストラクタで準備を整えます

draw() メソッドは、盤面 board の状態を画面に表示する処理を行っています

9 個の文字からなるリスト bd を用意して、board の値が 0 なら空きスロット、整数の 1 なら一人目の

プレイヤーの石、整数の 2 なら二人目のプレイヤーの石が置かれているとして、それぞれ文字'X' と'O' の文字を表示させています

print() 関数で、「文字列.format」という書き方によって、{} の位置に bd の文字を書いていますが、「文字列.format」という書き方よりも、「f 文字列」という書き方によって {} に値を埋め込む方が Pythonic な書き方だと「Effective Python」の項目 4 には書かれています

ソースコード 6.3　class Board

```
1  class Board:
2      def __init__(self):
3          self.board = [0,0,0,\
4                        0,0,0,\
5                        0,0,0]
6
7      def draw(self):
8          bd = ['', '', '',\
9                '', '', '',\
10               '', '', '']
11         n = 0
12         for val in self.board:
13             if val==0:
14                 bd[n] = str(n)
15             elif val==1:
16                 bd[n] = 'X'
17             elif val==2:
18                 bd[n] = 'O'
19             n += 1
20         print( '/---|---|---\\')
21         print( '| {} | {} | {} |'.format(bd[0], bd[1], bd[2]) )
22         print( '|---|---|---|')
23         print(f'| {bd[3]} | {bd[4]} | {bd[5]} |')
24         print( '|---|---|---|')
25         print(f'| {bd[6]} | {bd[7]} | {bd[8]} |')
26         print( '\\---|---|---/')
```

プレーヤのクラス：Player

Game クラスのコンストラクタの中で、Player クラスのオブジェクトとして、taro と hanako を生成しています。Player クラスのコンストラクタへの、一つ目の引数で名前を、二つ目の引数で盤面におく石に対応づけた整数を渡しています。

この整数は、taro と hanako の識別子としての役割を持たせますので、区別できる値でなければなりません。

また、game オブジェクトの中で PLAYER というリストを定義して、taro と hanako という Player クラスのオブジェクトのリストを作っています。PLAYER[0] によって taro オブジェクトを、PLAYER[1] によって hanako オブジェクトを取り出せることになります。

ゲームの進行は大まかにいうと、①盤面を表示して、②一方のプレーヤを選んで、③そのプレーヤが盤面上に石を置いて、④手番を交代して、をゲームオーバーまで繰り返すことになります

ソースコード 6.4　class Game

```
1  from Board import board
```

```
2  from Player import Player
3
4  class Game:
5      def __init__(self):
6          self.board = Board()
7          taro = Player('太郎', 1)
8          hanako = Player('花子', 2)
9          self.PLAYER = [taro, hanako]
10
11     def start(self):
12         turn = 0
13         while True:                          # 以下をゲームオーバーまで繰り返す
14             self.board.draw()                    # ①盤面を表示して
15             player = self.PLAYER[ turn ]         # ②プレーヤを選んで
16             player.put_stone( self.board )       # ③そのプレーヤが盤面に石を置いて
17             turn = (turn+1) % 2                  # ④手番を交代する
```

Player クラスでは、そのコンストラクタで、引数で受け取った名前と番号を各オブジェクトに保存するようにしています

put_stone() メソッドでは、input() 文でプレーヤにスロット番号の入力を促し、受け取った番号（文字列 s）を int(文字列) 関数によって整数に変換して、盤面 board 上のスロット番号の位置に、このプレーヤオブジェクトが保持している id 番号（1 か 2）を納めています

ソースコード 6.5　class Player

```
1  from Board import Board
2
3  class Player:
4      def __init__(self, name, id):
5          self.name = name
6          self.id = id
7
8      def put_stone(self, board):
9          s = input(self.name + "さんの手番です。スロット番号は？ : ")
10         n = int(s)
11         board.board[ n ] = self.id
```

勝敗の判定

Board クラスの winner() メソッドでは、行方向に 3 つ（row0, row1, row2）、列方向に 3 つ（col0, col1, col2）、斜め方向に 2 つ（crs0, crs1）のそれぞれで、各スロットの中の値が一致しているか否かをチェックしています

行または列、斜めのどれか一つでも一致しているものがあったら、そのときの player の勝ちです

ソースコード 6.6　class Board

```
1  class Board:
2      def __init__(self):
3          self.board = [0,0,0,0,0,0,0,0,0,0]
4
5      def draw(self):
6          bd = ['', '', '','', '', '','', '', '']
7          for n, val in enumerate( self.board ):
```

```
            ・
            変更なし
            ・
        print( '\\---|---|---/')

    # 以下を追加
    def winner(self, player):
        row0 = self.board[0]!=0 and\
               self.board[0]==self.board[1] and self.board[0]==self.board[2]
        row1 = self.board[3]!=0 and\
               self.board[3]==self.board[4] and self.board[3]==self.board[5]
        row2 = self.board[6]!=0 and\
               self.board[6]==self.board[7] and self.board[6]==self.board[8]
        col0 = self.board[0]!=0 and\
               self.board[0]==self.board[3] and self.board[0]==self.board[6]
        col1 = self.board[1]!=0 and\
               self.board[1]==self.board[4] and self.board[1]==self.board[7]
        col2 = self.board[2]!=0 and\
               self.board[2]==self.board[5] and self.board[2]==self.board[8]
        crs0 = self.board[0]!=0 and\
               self.board[0]==self.board[4] and self.board[0]==self.board[8]
        crs1 = self.board[2]!=0 and\
               self.board[2]==self.board[4] and self.board[2]==self.board[6]
        if col0 or col1 or col2 or row0 or row1 or row2 or crs0 or crs1:
            return True
```

勝敗を判定して結果を表示し、break によってゲームのループを抜けます

ソースコード 6.7　class Game

```
from Board import Board
from Player import Player

class Game:
    def __init__(self):
        self.board = Board()
        taro = Player('太郎', 1)
        hanako = Player('花子', 2)
        self.PLAYER = [taro, hanako]

    def start(self):
        turn = 0
        while True:
            self.board.draw()
            player = self.PLAYER[turn]

            b1 = self.board.winner()
            if b1:
                print( player.name + 'さん　の勝ち')
                break

            turn = (turn+1) % 2
```

引き分けの判定

board オブジェクトが持っている board プロパティは、9 個の各スロットの状態（空かマルかバツか→ 0 か 1 か 2 か）を保持するリストでした。スロットが空き（スロットの値がゼロ）であれば、石を置くこ

とができます。

値がゼロのスロット番号（n）を集めたリストを vacant に作り、その長さは len() 関数で調べることができます。リストの長さがゼロであるということは、空のスロット（値がゼロのスロット）は無くなったということですから、もう石を置くことはできません（このとき引き分けです）。

```
        # 引き分けの判定はここから
        empty = []
        n = 0
        for slot in board.board:
            if slot==0:
                empty.append( n )
            n += 1
        if len( empty )==0:
            return False
        # ここまで
```

```
    # 引き分けの判定はここから
    empty = []
    for n in range( len(board.board) ):
        slot = board.board[n]
        if slot==0:
            empty.append( n )

    if len( empty )==0:
        return False
    # ここまで
```

引き分けの判定部分は、上のように書くこともできますが、ここでは以下のような enumerate() 関数の使い方を学習しましょう。「Effective Python」には項目 7 で、range ではなく enumerate を使うのが Pythonic だとされています。

また、if len(empty)==0: という記述は、if not empty: と書いた方が Pythonic だと同じ本の第一章には書かれています。

ソースコード 6.8　class Board

```
class Board:
    def __init__(self):
        self.board = [0,0,0,0,0,0,0,0,0]

    def draw(self):
        bd = ['', '', '','', '', '','', '', '']
        for n, val in enumerate( self.board ):
            ・
            変更なし
            ・
        print( '\\---|---|---/')

    def winner(self, player):
            ・
            変更なし
            ・
        if col0 or col1 or col2 or row0 or row1 or row2 or crs0 or crs1:
            return True

    # 以下を追加
    def vacant(self):
        empty = []
        for n, slot in enumerate(self.board):
            if slot == 0:
                empty.append(n)
        return empty
```

入力されたスロット番号 n が、vacant リストの中にあることを確認すること（if n in vacant:）によって、既に置かれた石の上に上書きすることを防いでいます。

今の段階で、put_stone() メソッドが返す値は、False ならば引き分け、True ならばボード上に石を置きましたという意味になります

ソースコード 6.9　class Player

```
from Board import Board

class Player:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def put_stone(self, board):
        # 引き分けの判定はここから
        vacant = board.vacant()
        if not vacant:
          return False
        # ここまで
        while True:
            s = input(self.name + "さんの手番です。スロット番号は？ : ")
            n = int(s)
            if n in vacant:       # 空きスロットのリストに番号 n は含まれているか？
                board.board[ n ] = self.id
                break
        return True
```

Game クラスでは、put_stone() メソッドを呼び出して判定する必要があります

引き分けの判定（b2）で False が返された（not b2 == True）なら、’引き分け’ と表示してゲームの反復を抜けます（break）

b2 が True を返してきた場合は、順当にその時の手番（turn）の player オブジェクトが、盤面オブジェクト（self.board）に石を置いた（put_stone() した）場合になりますから、そのままゲームを続行します

引き分けと勝敗の判定を、ゲーム進行の途中に入れています

引き分けた場合、あるいは勝敗のついた場合のいずれの場合でも、break 文によってゲームの進行（反復）を終えています

ソースコード 6.10　class Game

```
from Board import Game
from Player import Player

class Game:
    def __init__(self):
        self.board = Board()
        taro = Player('太郎', 1)
        hanako = Player('花子', 2)
        self.PLAYER = [taro, hanako]

    def start(self):
        turn = 0
        while True:
            self.board.draw()
            player = self.PLAYER[turn]

            b2 = player.put_stone( self.board )
```

```
18              if not b2:
19                  print('引き分け')
20                  break
21
22              b1 = self.board.winner()
23              if b1:
24                  print( player.name + 'さん　の勝ち')
25                  break
26
27              turn = (turn+1) % 2
```

6.1.2　太郎さんとコンピュータの対戦

花子さんをコンピュータに変更します

Game クラスの最初、コンストラクタで Player クラスのオブジェクト生成を変更します

ソースコード 6.11　class Game

```
1  from Board import Game
2  from Player import Player
3
4  class Game:
5      def __init__(self):
6          self.board = Board()
7          human = Player('Taro', 1)          # 変更点はここから3行
8          machine = Player('computer', 2)
9          self.PLAYER = [human, machine]
10
11     def start(self):
12         turn = 0
13         while True:
14             self.board.draw()
15                 ・
16                 ・    この部分に変更なし
17                 ・
```

Player クラスでは、computer オブジェクトの場合には乱数によって、 vacant リストの中から石をおくスロット番号を選ぶようにします

human オブジェクトの場合は、スロット番号の入力を促しています

Player クラスのオブジェクトは、human の場合と machine の場合がありますから、Player クラスのそれぞれのオブジェクトが持っている self.id の値を見て、それが 1 なら human オブジェクト、2 なら machine オブジェクトだと判定している部分に注目しましょう

ソースコード 6.12　class Player

```
1  from Board import Board
2
3  class Player:
4      def __init__(self, name, id):
5          self.name = name
6          self.id = id
7
8      def put_stone(self, board):
```

```
9           # 引き分けの判定はここから
10          vacant = board.vacant()
11          if not vacant:
12              return False
13          # 変更点は以下の部分
14          if self.id == 2:        # Computerの場合は乱数で
15              n = randint( 0, len(vacant)-1 )
16              board.board[ vacant[n] ] = self.id
17          else:                   #
                Taroの場合はキーボード入力でスロット番号を指定させる
18              while True:
19                  s = input(self.name + "さんの手番です。スロット番号は？ : ")
20                  n = int(s)
21                  if n in vacant:
22                      board.board[ n ] = self.id
23                      break
24          return True
```

winner() メソッドをもう少し簡潔に記述し直すことにしましょう

ソースコード 6.13　class Board

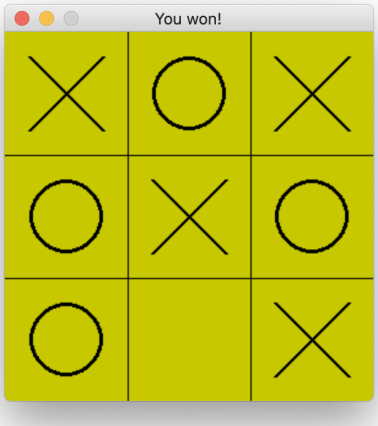
```
1   class Board:
2           .
3           . この部分に変更なし
4           .
5       def winner(self, player):
6           def scan(n1, n2, n3):
7               return self.board[n1]!=0 and\
8                      self.board[n1]==self.board[n2] and\
9                      self.board[n1]==self.board[n3]:
10          return scan(0,1,2) or scan(3,4,5) or scan(6,7,8) or\
11                 scan(0,3,6) or scan(1,4,7) or scan(2,5,8) or\
12                 scan(0,4,8) or scan(2,4,6)
```

6.2　GUI(Graphical User Interface) 版

ここまでは CUI（Character User Interface）ベースのゲームでしたが、ここからは、Pygame を使って GUI（Graphical User Interface）版のゲームを作っていきます

新しいプロジェクトを開始することにしましょう

まずは、コンピュータが乱数で石を置いてくる単純な方法を実現します



Params.py

今回は、Params.py というファイルに、Params クラスと State クラスという、2 つのクラスを定義しておきます

ソースコード 6.14　Params.py

```
1  from enum import auto
2
3  class Params:
4      HUMAN_ID = auto()
5      MACHINE_ID = auto()
6      EMPTY_ID = auto()
7
8  class State:
9      GAME = auto()
10     MACHINE_WIN = auto()
11     HUMAN_WIN = auto()
12     DRAW = auto()
```

Params クラスには、盤面に置く石の識別情報、HUMAN_ID と MACHINE_ID、及び空きスロットを識別する EMPTY_ID の 3 つの識別子を持たせます

State クラスには、盤面の状態＝ゲームの進行状況を示す定数、即ち、GAME はゲーム進行中の状態、HUMAN_WIN は人が勝った状態、MACHINE_WIN はコンピュータが勝った状態、DRAW は引き分けの状態を表す定数として保持させます

いずれのクラスも、auto() という関数を使って自動的に重複しない値を持たせることができています

auto() によって自動的に区別された値が割り振られるため、プログラム中で変数名を参照していく以上、それらの値そのものがどのような値なのかは知らなくても問題はありませんね

また、これらはクラス変数と呼ばれ、「クラス名. 変数名」という形で使うことができて、このクラスから導出されるオブジェクトでは共通に持っている変数になります

オブジェクト毎に持たせているオブジェクト変数「self. 変数名」とは別の変数ですので、区別して理解するようにしましょう

Screen クラス

GUI にするので、pygame で Window 画面を用意するための Screen クラスを作ります

ソースコード 6.15　class Screen

```
import pygame

class Screen:
    def __init__(self, wh=(600,600), bgcolor=(0,0,0)):
        self.WIDTH = wh[0]
        self.HEIGHT = wh[1]
        self.COLOR = bgcolor
        self.SIZE = (self.WIDTH, self.HEIGHT)
        self.surface = pygame.display.set_mode(self.SIZE)
        self.SLOTW = self.WIDTH//3
        self.SLOTH = self.HEIGHT//3

    def fill(self):
        self.surface.fill(self.COLOR)
        BLACK = (0,0,0)
        for i in range(1,3):
            startp = (0, self.SLOTH*i)
            endp = (self.WIDTH, self.SLOTH*i)
            pygame.draw.line(self.surface, BLACK, startp, endp)
            startp = (self.SLOTW*i, 0)
            endp = (self.SLOTW*i, self.HEIGHT)
            pygame.draw.line(self.surface, BLACK, startp, endp)

    def caption(self, str):
        pygame.display.set_caption(str)
```

コンストラクタへの引数で、盤面のサイズと背景色を受け取ることができますが、特に指定しない場合にはデフォルト値が用いられる様にしています

三目並べは 3 x 3 のスロットを持つ盤面ですので、そのための描画を fill() メソッドで行っています

caption() メソッドでは、引数で受け取った文字列を Window のタイトルバーに表示できるようにしています

Board クラス

Board クラスは、Screen クラスを継承させるようにしていますので、Screen クラスもプロパティやメソッドを、あたかも Board クラスで用意したかのように使うことができます

Board クラスのコンストラクタでは、引数で盤面のサイズ、背景色、キャプションに表示する文字列を受け取ることができるようにしていますが、時に指定しなかった場合にデフォルト値が使われるように記述しています

Board クラスのコンストラクタの冒頭で、Screen クラスのコンストラクタを呼び出して、Screen の初期化を実行し、キャプションの表示、そして盤面の実態である 3 x 3 = 9 つの EMPTY 要素を持つ board という名のリストを用意しています

また、ゲームの状態を self.state に持たせることとし、ゲーム進行中の状態 GAME を初期状態として代入しています

ソースコード 6.16　class Board

```
import pygame
from Screen import Screen
from Params import State,Params

class Board( Screen ):
    def __init__(self, wh=(300,300), color=(200,200,0), str='Tic-Tac-Toe'):
        super().__init__(wh, color)
        self.caption(str)
        self.board = [Params.EMPTY_ID for _ in range(9)]
        self.state = State.GAME

    def draw(self):
        self.fill()
        N = 3
        RADIUS = int(self.SLOTW*0.2)
        BLACK = (0,0,0)
        for ypos in range(N):
            yc = ypos * self.SLOTH + self.SLOTH // 2
            for xpos in range(N):
                xc = xpos*self.SLOTW + self.SLOTW // 2
                i = ypos * N + xpos
                if self.board[i] == Params.HUMAN_ID:
                    pygame.draw.circle(self.surface, BLACK, (xc,yc), RADIUS, 3)
                elif  self.board[i] == Params.MACHINE_ID:
                    startp = (xc - RADIUS, yc - RADIUS)
                    endp = (xc + RADIUS, yc + RADIUS)
                    pygame.draw.line(self.surface, BLACK, startp, endp, 3)
                    startp = (xc + RADIUS, yc - RADIUS)
                    endp = (xc - RADIUS, yc + RADIUS)
                    pygame.draw.line(self.surface, BLACK, startp, endp, 3)

    def winner(self):
        def scan(n1, n2, n3):
            p = self.board[n1] != 0 and \
                self.board[n1] == self.board[n2] and \
                self.board[n1] == self.board[n3]
            if p:
                if self.board[n1]==Params.HUMAN_ID:
                    self.state = State.HUMAN_WIN
                elif self.board[n1]==Params.MACHINE_ID:
                    self.state = State.MACHINE_WIN
            else:
                if not self.vacant():
                    self.state = State.DRAW
                else:
                    self.state = State.GAME
            return p
        #
        return scan(0, 1, 2) or scan(3, 4, 5) or scan(6, 7, 8) or \
               scan(0, 3, 6) or scan(1, 4, 7) or scan(2, 5, 8) or \
               scan(0, 4, 8) or scan(2, 4, 6)

    def vacant(self):
        empty = []
        for n, slot in enumerate(self.board):
            if slot == Params.EMPTY_ID:
```

```
57                 empty.append(n)
58         return empty
```

draw() メソッドは、盤面 board リストの状態を Window の Screen 上に描画しています

board リストの要素が HUMAN_ID と一致するなら draw.circle() で○を描き、MACHINE_ID と一致するなら draw.line() を使って×を描いています

winner() メソッドでは、ゲームの勝敗、引き分け、ゲーム進行中の判別をして、self.state プロパティにそれぞれの値を設定しています

winner() メソッドの中に持っているローカルな scan() 関数が受け取る 3 つのスリット番号を見て、board リストの該当するスロットに入っている石が同じ識別子かどうかを判定しています

同じだったら、その石が HUMAN_ID なのか MACHINE_ID なのかを判別して、勝者を self.state に設定します

違っていたら、石を置く場所が残っているかどうかを調べ、残っていないないなら引き分け、残っているようならゲーム継続の状態だとしています

scan() が返す値が True なら、winner の返す値も True になるので、このとき勝敗がついたという意味になります

vacant() メソッドは、CUI でも説明した空きスロットのリストを返すメソッドです

Machine クラス

Machine クラスの put_stone() メソッドでは、まず、空きスロットのリストを取得し、そのリスト内の要素を乱数で選んで、board リスト上のその位置に、MACHINE_ID を代入しています

True を返した場合は石を普通に置いた場合、False を返すのは盤面に空きがなかった場合です

ソースコード 6.17　class Machine

```
1  from random import randint
2  from Params import Params
3
4  class Machine():
5      def __init__(self, name):
6          self.name = name
7
8      def put_stone(self, board):
9          vacant = board.vacant()
10         if not vacant:
11             return False
12         n = randint(0, len(vacant)-1)
13         board.board[ vacant[n] ] = Params.MACHINE_ID
14         return True
```

Human クラス

Human クラスの put_stone() メソッドでは、まず空きスロットのリストを取得し、空きがないなら False で戻ります

空きがある場合は、put_stone() メソッドの中に、ローカルに定義した get_slot() 関数を呼びだし、その引数に、マウスがクリックした盤面の座標をタプルの形で渡します

get_slot() 関数は、board やスロットのサイズを元に、マウスがクリックしたのがどのスロット上なの

かを求めて、そのスロット番号を返しています

戻されたスロット番号が、空きスロットのリストに含まれていたなら、board のその場所に HUMAN_ID を代入して True で戻ります

マウスでクリックしたスロット番号が、空きスロットリストに含まれていないなら、それは、既に他の石が置かれているスロットだと言うことになりますから、もう一度クリックでスロットを選び直して下さいという意味で、None を返しています

ソースコード 6.18　class Human

```
from Params import Params

class Human:
    def __init__(self, name):
        self.name = name

    def put_stone(self, board, pos=(-1,-1)):
        def get_slot(xy):
            x, y = xy[0], xy[1]
            N = 3
            for ypos in range(N):
                y0 = board.SLOTH * ypos
                y1 = board.SLOTH * (ypos + 1)
                for xpos in range(N):
                    x0 = board.SLOTW * xpos
                    x1 = board.SLOTW * (xpos + 1)
                    if y0 < y < y1 and x0 < x < x1:
                        return ypos * N + xpos
            return -1
        vacant = board.vacant()
        if not vacant:
            return False
        n = get_slot(pos)
        if n in vacant:
            board.board[n] = Params.HUMAN_ID
        else:
            return None
        return True
```

Game クラス

Game クラスのコンストラクタでは、引数 turn に文字列を受け取り、それによって先手と後手を決めています

change_tuen() メソッドは、手番を交代します

fine() メソッドは、このアプリケーションを終了させるときのメソッドで、Window 上の QUIT のエベントを受け取ったときに呼び出されます

キーボードイベントの取得部分で、MOUSEBUTTONUP を拾っており、human オブジェクトの put_stone() メソッドにクリックした座標 event.pos を渡しています

judge() メソッドは、盤面の状態 board オブジェクトの state プロパティの値を見て、勝敗が決したのか、引き分けなのか、ゲーム思考の継続なのかを判定し、ゲーム進行中の時だけ True を、そうでない時は False を返しています

start() メソッドはゲームの進行そのもので、以下を繰り返しています

①キーイベントをチェックして HUMAN の手番ならこの中で HUMAN がスロットを選ぶことになります②盤面の状態を描画します③ MACHINE の手番なら、machine オブジェクトの put_stone() メソッドを呼び出しています④勝敗の判定をして、⑤画面を更新します

ソースコード 6.19　class Game

```
import sys
import pygame
from pygame.locals import QUIT, MOUSEBUTTONUP
from Board import Board
from Human import Human
from Machine import Machine
from Params import State

class Game:
    def __init__(self, turn='human'):
        pygame.init()
        self.board = Board()
        self.human = Human('Taro')
        self.machine = Machine('Computer')
        self.turn = False if turn=='human' else True
        self.clock = pygame.time.Clock()
        self.FPS = 10

    def change_turn(self):
        self.turn = not self.turn

    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 == MOUSEBUTTONUP:
                if not self.turn:
                    p = self.human.put_stone(self.board, event.pos)
                    if p:
                        self.change_turn()
                    elif p==False:
                        pass    # DRAW
                    else:
                        pass

    def judge(self):
        result = False
        if self.board.winner():
            if self.board.state == State.MACHINE_WIN:
                self.board.caption('Computer won the game!')
            elif self.board.state == State.HUMAN_WIN:
                self.board.caption('You won!')
        else:
            if self.board.state == State.DRAW:
                self.board.caption('Draw!')
            elif self.board.state == State.GAME:
                result = True
```

```
52          return result
53
54      def start(self):
55          while True:
56              self.key_event()
57              self.board.draw()
58              if self.turn:
59                  if self.machine.put_stone(self.board):
60                      self.change_turn()
61              self.judge()
62              pygame.display.update()
63              self.clock.tick(self.FPS)
```

main.py

先手が誰なのかを、Game クラスのコンストラクタに指示しています

ソースコード 6.20　main.py

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

6.3　MiniMax 法

ここまでコンピュータが採用してきた戦略は、「空いているスロットの中から乱数で選んだ場所に石を置く」という単純なものでした

ここでは、絶対に負けないコンピュータ、（人が最善の手をとった場合に引き分けたとしても負けることはありません）を実現していきます

6.3.1　main の変更

Game クラスのインスタンス game を生成する際、コンストラクタへの引数で、先手は'human' か'machine' か、コンピュータの採る戦略は乱数'random' か'minimax' かを指定するように直します

ソースコード 6.21　main.py

```
1  from Game import Game
2  from Params import Params
3
4  if __name__ == '__main__':
5      sente = 'machine'            # 'machine' or 'human'
6      senryaku = 'minimax'         # 'random' or 'minimax'
7      game = Game( turn=sente, strategy=senryaku )
8      game.start()
```

6.3.2　Game クラスの変更

Game クラスのコンストラクタにおいて、MACHINE が採る戦略を引数 strategy で受け取り、それを Machine クラスのコンストラクタに渡しています

ソースコード 6.22　Game.py

```
1  import sys
2  import pygame
3  from pygame.locals import QUIT, MOUSEBUTTONUP
4  from Board import Board
5  from Human import Human
6  from Machine import Machine
7  from Params import State
8
9  class Game:
10     def __init__(self, turn='human', strategy='random'):
11         pygame.init()
12         self.board = Board()
13         self.human = Human('Taro')
14         self.machine = Machine('Computer', strategy)
15         self.turn = False if turn=='human' else True
16         self.clock = pygame.time.Clock()
17         self.FPS = 10
18     ・
19     以下に変更なし
20     ・
21     ・
```

6.3.3　Board クラスの変更

次の 2 つのメソッドは、MiniMax 戦略で使うために追加します
can_put() メソッドは、引数で受け取った番号のスロットが空いているかどうかを返します
undo() メソッドは、一旦石を置いたスロットを、元の空きスロットに戻すメソッドです
デバッグのために用意したメソッドです（完成したら不要です）
debug_board() メソッドは、デバッグ用に現在の盤面の状態をファイル出力するメソッドです

ソースコード 6.23　Board.py

```
1  import pygame
2  from Screen import Screen
3  from Params import State,Params
4
5  class Board( Screen ):
6      def __init__(self, wh=(300,300), color=(200,200,0), str='Tic-Tac-Toe'):
7          ・
8          変更なし
9          ・
10         ・
11     def draw(self):
12         ・
13         変更なし
```

```
        ・
        ・
    def winner(self):
        ・
        変更なし
        ・
        ・
    def vacant(self):
        ・
        変更なし
        ・
        ・
    # 以下を追加
    def undo(self, n):
        self.board[n] = Params.EMPTY_ID

    def can_put(self, n):
        if self.board[n]==Params.EMPTY_ID:
            return True
        return False

    # ボードを表示する for Debug
    def debug_board(self, seq, debugstring):
        str0 = '\n('+str(seq)+')'
        tmp = []
        for i in range(9):
            if self.board[i] == Params.EMPTY_ID:
                tmp.append(' ')
            elif self.board[i] == Params.HUMAN_ID:
                tmp.append('o')
            elif self.board[i] == Params.MACHINE_ID:
                tmp.append('x')
        str1 =  '\n{0[0]}|{0[1]}|{0[2]}\t<-----\t0|1|2'\
                '\n{0[3]}|{0[4]}|{0[5]}\t<-----\t3|4|5'\
                '\n{0[6]}|{0[7]}|{0[8]}\t<-----\t6|7|8\n'.format(tmp)
        str2 = str0 + str1 + debugstring + '\n'
        with open('textfile.txt', 'a', encoding='utf-8') as f:
            f.write(str2)
```

6.3.4　Machine クラスの変更

Machine クラスのコンストラクタで、戦略が'random' か'minimax' かを受け取っています

Strategy クラスを継承しており、MiniMax の具体的な戦略のプログラムコードは上位クラスの Strategy に記述することにします

put_stone() メソッドでは、①空きスロットのリスト vacant を取得し、②空きがないなら③石を置けないので False で返ります。④ strategy が'random' だった場合、⑤乱数で vacant のインデックス n を選び、⑥ vacant の n 番目にあるスロット番号に石を置きます。⑦ strategy が'minimax' だった場合、⑧ MiniMax 法によるベストな手をスロット番号 n で受け取り、⑨そこに石を置きます

ソースコード 6.24　Machine.py

```
from random import randint
from Params import Params
from Strategy import Strategy
```

```
4
5  class Machine( Strategy ):
6      def __init__(self, name, strategy):
7          super().__init__()
8          self.name = name
9          self.strategy = strategy
10
11     def put_stone(self, board):
12         vacant = board.vacant()
13         if not vacant:
14             return False
15         if self.strategy=='random':
16             n = randint(0, len(vacant)-1)
17             board.board[ vacant[n] ] = Params.MACHINE_ID
18         elif self.strategy=='minimax':
19             n = self.bestMove(board, vacant)
20             board.board[ n ] = Params.MACHINE_ID
21         return True
```

6.3.5　Strategy クラスを追加

ここに MiniMax 戦略のコードをまとめて記述します

bestMove() メソッドでは、①引数で空のスロット番号のリスト vacant を受け取り、②そのリストの要素の全てについて③順に試して④その手を評価していきます（Maximizer である Machine が③で石を置いた直後ですから、次は Human 即ち Minimizer の手番ですよ、という意味で False を引数に指示しています）⑤高い評価値の得られたスロット番号は、⑥ bestMove の候補として更新していきます⑦③で選んだ手を元の空のスロットに戻して、③次の手を置いてみて④評価することを繰り返します

minimax() メソッドでは、①まず、board インスタンスの winner() メソッドを呼んで現在の盤面の状態が、ゲーム進行中（GAME）なのか、勝敗が決した（MACHINE_WIN か HUMAN_WIN）のか、引き分け（DRAW）だったのかを判定します②ゲーム進行中（GAME）でないなら、③評価関数であるローカルな evaluate() メソッドを呼びだし、結果を返します④ゲーム進行中ならば、⑤全てのスロット番号を順に選んで、⑥そのスロットが空で石を置けるかどうかを判定し、置けるならば、⑦そのスロットに石を置きます（Maximizer=Machine=’×’ か、Minimizer=Human=’○’ かによって置く石は違います）⑧再帰的に minimax() メソッドを呼び出します（isMaximizingPlayer を not として引数に指示し、ターンを交互に割り当てるようにしています）⑨ bestVal を更新していきます。⑩石を置いた番号のスロットを空に戻します

minimax() メソッドの中に定義された評価関数の evaluate() は、ゲームが進行状態（GAME）でない場合に呼び出されるので、Machine が勝った場合（MACHINE_WIN）、Human が勝った場合（HUMAN_WIN）、引き分けだった場合（DRAW）に、それぞれ評価値を返しています

ソースコード 6.25　Strategy.py

```
1  from Params import Params, State
2
3  class Strategy:
4      INFINITY = 10000
5      def __init__(self):
6          pass
7
```

```
8       def bestMove(self, board, vacant):
9           bestEval = -Strategy.INFINITY
10          bestMove = -1
11          self.debug_seq = 0
12          for n in vacant:
13              board.board[n] = Params.MACHINE_ID
14              eval = self.minimax(board, 0, False)
15              if bestEval<eval:
16                  bestEval = eval
17                  bestMove = n
18              board.undo(n)
19          return bestMove
20
21      def minimax(self, board, depth, isMaximizingPlayer):
22          def evaluate(depth):
23              if board.state == State.MACHINE_WIN:
24                  board.state = State.GAME
25                  return 10-depth      #return 10
26              elif board.state == State.HUMAN_WIN:
27                  board.state = State.GAME
28                  return depth-10      #return -10
29              else:
30                  self.state = State.GAME
31                  return 0
32
33          board.winner()
34          if board.state != State.GAME:
35              return evaluate(depth)
36          bestVal = -Strategy.INFINITY if isMaximizingPlayer else +Strategy.INFINITY
37          for n in range(9):
38              if board.can_put(n):
39                  board.board[n] = Params.MACHINE_ID if isMaximizingPlayer else
                        Params.HUMAN_ID
40                  value = self.minimax(board, depth + 1, not isMaximizingPlayer)
41                  bestVal = max(value, bestVal) if isMaximizingPlayer else min(value,
                         bestVal)
42                  board.undo(n)
43          return bestVal
```

6.4　Alpha Bata Pruning

Alpha Beta Pruning は新しい手法のアルゴリズムではなく、MiniMax 法を最適化したものにすぎません

MiniMax 法は、勝敗が決着するまで全ての手を試行して、その上で最善手を選ぶため、引き分けることはあっても負けることはありません。

しかし、全ての手を試行するというのは、多くのゲームで現実的な方法にはなりません。三目並べの様な小規模のゲームでは、それほど問題になりませんが、リバーシなどになると思考時間がかかりすぎて実用的ではありません

そこで、MiniMax 法の試行を途中で合理的に打ち切る手法がアルファ刈りベータ刈りと呼ばれる手法です

6.4.1　main の変更

Game クラスのインスタンス game を生成する際、コンストラクタへの引数で、先手は'human' か'machine' か、コンピュータの採る戦略は乱数'random' か'minimax' か'alphabeta' の何れかを指定するようにします

ソースコード 6.26　main.py

```
from Game import Game
from Params import Params

if __name__ == '__main__':
    sente = 'human'            # 'machine' or 'human'
    senryaku = 'alphabeta'         # 'random', 'minimax' or 'alphabeta'
    game = Game( turn=sente, strategy=senryaku )
    game.start()
```

6.4.2　Machine クラスの変更

戦略として、alphabeta を選べるように直します

ソースコード 6.27　class Machine

```
from random import randint
from Params import Params
from Strategy import Strategy

class Machine( Strategy ):
    def __init__(self, name, strategy):
        super().__init__()
        self.name = name
        self.strategy = strategy

    def put_stone(self, board):
        vacant = board.vacant()
        if not vacant:
            return False
        if self.strategy=='random':
            n = randint(0, len(vacant)-1)
            board.board[ vacant[n] ] = Params.MACHINE_ID
        elif self.strategy=='minimax':
            n = self.bestMove(board, vacant)
            board.board[ n ] = Params.MACHINE_ID
        elif self.strategy=='alphabeta':
            n = self.bestMoveAB(board, vacant)
            board.board[ n ] = Params.MACHINE_ID
        return True
```

6.4.3　Strategy クラスの変更

具体的な戦略は、こちらのクラスに記述しています

ソースコード 6.28　class Strategy

```
from Params import Params, State

class Strategy:
    INFINITY = 10000
    def __init__(self):
        pass

    def bestMove(self, board, vacant):
            ・
            変更なし
            ・
            ・

    def minimax(self, board, depth, isMaximizingPlayer):
            ・
            変更なし
            ・
            ・

    def bestMoveAB(self, board, vacant):
        bestEval = -Strategy.INFINITY
        bestMove = -1
        for n in vacant:
            board.board[n] = Params.MACHINE_ID
            eval = self.minimaxab(board, 0, 0, False, -Strategy.INFINITY, +Strategy
                .INFINITY)
            if bestEval<eval:
                bestEval = eval
                bestMove = n
            board.undo(n)
        return bestMove

    def minimaxab(self, board, node, depth, isMaximizingPlayer, alpha, beta):
        def evaluate(depth):
            if board.state == State.MACHINE_WIN:
                board.state = State.GAME
                return 10-depth     #return 10
            elif board.state == State.HUMAN_WIN:
                board.state = State.GAME
                return depth-10     #return -10
            else:
                self.state = State.GAME
                return 0

        board.winner()
        if board.state != State.GAME:
            return evaluate(depth)

        bestVal = -Strategy.INFINITY if isMaximizingPlayer else +Strategy.INFINITY
        for n in range(9):
            if board.can_put(n):
                board.board[n] = Params.MACHINE_ID if isMaximizingPlayer else
                    Params.HUMAN_ID
                value = self.minimaxab(board, node+1, depth+1, not
                    isMaximizingPlayer, alpha, beta)
                board.undo(n)
```

```
54                  if isMaximizingPlayer:
55                      bestVal = max(value, bestVal)
56                      alpha = max(alpha, bestVal)
57                  else:
58                      bestVal = min(value, bestVal)
59                      beta = min(beta, bestVal)
60                  if beta<=alpha:
61                      #print('depth=',depth)
62                      break
63          return bestVal
```

第 7 章

リバーシ（オセロ）：N × N

8 × 8 サイズのゲームが一般的ですが、6 × 6 盤の縮小リバーシでは必勝法が見つかっているようです
1993 年にイギリスの Joel Feinstein が、6 × 6 盤の縮小リバーシで、先手が最善手を尽くしたとしても後手が「20 対 16」で勝つ、(先手 16 対、後手 20 で、後手の 4 目勝ち) という事を示したということです
ここでは、4 × 4「以上」の大きさの盤面に対応するリバーシを作ります。あなたにゲームを最後まで続ける根気があるなら、10 × 10 でも 20 × 20 でもゲームすることができるかもしれません

7.1　CUI 版

以下のゲーム例では、×（BLACK）の手番ですが、○（WHITE）を裏返せないところに×を置こうとしたり、×（BLACK）を置ける場所があるのに pass を宣言したりすると、再入力を促されています
○（WHITE）の手番で、まずは乱数によってコンピュータが石を置くスロットを決めていますが、後ほど Strategy クラスで、コンピュータの強い戦略について考察していきます

```
WHITE:2, BLACK:2
  0 1 2 3 →x
0 . . . .
1 . X O .
2 . O X .
3 . . . .
↓
y
Your turn. [yx] or "pass": 23
Your turn. [yx] or "pass": pass
Your turn. [yx] or "pass": 13

WHITE:1, BLACK:4
  0 1 2 3 →x
0 . . . .
1 . X X X
2 . O X .
3 . . . .
↓
y
WHITE:3, BLACK:3
  0 1 2 3 →x
0 . . . .
1 . X X X
```

```
2 . O O O
3 . . . .
↓
y
Your turn. [yx] or "pass": 33

WHITE:1, BLACK:6
  0 1 2 3 →x
0 . . . .
1 . X X X
2 . O X X
3 . . . X
↓
y
WHITE:3, BLACK:5
  0 1 2 3 →x
0 . O . .
1 . O X X
2 . O X X
3 . . . X
↓
y
Your turn. [yx] or "pass": 00
```

（　・・　途中略　・・　）

```
WHITE:5, BLACK:9
  0 1 2 3 →x
0 X X X O
1 O X X X
2 O O X X
3 O . . X
↓
y
Your turn. [yx] or "pass": 31

WHITE:4, BLACK:11
  0 1 2 3 →x
0 X X X O
1 O X X X
2 O X X X
3 O X . X
↓
y
WHITE:7, BLACK:9
  0 1 2 3 →x
0 X X X O
1 O X X X
2 O O X X
3 O O O X
↓
y

Winner : BLACK
```

7.1.1　Human vs. Machine

Game クラス

メインでは、Game クラスのコンストラクタでへの引数で盤面のサイズ（N=4 以上の偶数）、先手を'human', 'machine' から、戦略を'random', 'maxflip', 'minimax', 'alphabeta' から指定します

ソースコード 7.1　main

```
from Game import Game

if __name__ == '__main__':
    sente = 'human'          # 'human, 'machine'
    senryaku = 'maxflip'     # 'random', 'maxflip', 'minimax', 'alphabeta'
    game = Game( N=6, turn=sente, strategy=senryaku )       # board size -> N x N
    game.start()
```

Game クラスのコンストラクタで、盤面クラス（Board）のインスタンス（board）、人クラス（Human）のインスタンス（human）、コンピュータクラス（Machine）のインスタンス（machine）を生成し、インスタンス変数 self.player には、human と machine の 2 つのインスタンスを要素とするリストを用意しています

start() メソッドで、ゲームの進行の主要部分を記述しています

①盤面を印刷（board.print()）し、②プレーヤの打つ手を選び（player[turn].Te(self.board)）、③もし、パス（Board.PASS）でなかった場合は、④盤面に石を置く（board.putStone(te,color)）ことになります。⑤パスならば盤面に石を置かずに、プレーヤを交代（turn = (turn+1)%2）します

⑥盤面の状況から勝敗の判定 (board.Winner()) を行い、戻り値が Board.GAME ならばゲームを継続、それ以外（BLACK_WIN か WHITE_WIN か DRAW）ならば、ゲームを終了して結果の勝敗を表示します

ソースコード 7.2　Game class

```
from Board import Board
from Human import Human
from Machine import Machine
from Stone import Stone

class Game():
    def __init__(self, N=4, turn='human', strategy='random'):
        self.board = Board(NW=N)
        human = Human(color=Stone.BLACK)
        machine = Machine(color=Stone.WHITE, strategy=strategy)
        self.player = [human, machine]
        self.turn = 0
        if turn=='machine':
            self.turn = 1

    def start(self):
        turn = self.turn
        winner = Board.GAME
        while winner==Board.GAME:
            self.board.print()                    # ①

```

```
22              te = self.player[turn].Te(self.board) # ②
23              if te!=Board.PASS:                    # ③
24                  color = self.player[turn].color
25                  self.board.put_stone(te,color)    # ④
26
27              turn = (turn+1)%2                     # ⑤
28              winner = self.board.winner()          # ⑥
29
30          self.board.print()
31          print()
32          if winner==Board.BLACK_WIN:
33              print('Winner : BLACK')
34          elif winner==Board.WHITE_WIN:
35              print('Winner : WHITE')
36          elif winner==Board.DRAW:
37              print('DRAW')
```

Stone クラス

石のクラスでは、BLACK、WHITE、EMPTY の中の何れかの色特性（self.color）、及び盤面上の位置（self.locate）をプロパティとして持たせています

用意しているメソッドは、各プロパティに関するセッターやゲッターです

ソースコード 7.3　Stone class

```
1  class Stone():
2      EMPTY = 0
3      BLACK = 1
4      WHITE = 3 - BLACK
5      MACHINE_ID = WHITE
6      HUMAN_ID = BLACK
7      def __init__(self, te, color):
8          self.color = color
9          self.locate = te
10
11     def getLocate(self):
12         return self.locate
13
14     def setColor(self, color):
15         self.color = color
16
17     def getColor(self):
18         return self.color
19
20     def flipColor(self):
21         self.color = 3 - self.color
22
23     def getFigure(self):
24         if self.color==Stone.BLACK:
25             return 'X'
26         elif  self.color==Stone.WHITE:
27             return 'O'
28         return '.'
```

Board クラス

このクラスでは、盤面の状態をリスト（self.stones）に保持しています

コンストラクタでは、まず引数で示されたサイズ（NW * NW）の空の（Stone.EMPTY）盤面を用意しています

次に、盤面の中央に BLACK と WHITE の石をそれぞれ 2 個ずつ配置しますが、配置のパタンは乱数によって決めています

石の数を nWhite,nBlack,nEmpty に保持させ、winner() メソッドの最初でそれぞれの値を数えています

nWhite + nBlack + nEmpty は盤面のスロットの総数になりますから、nEmpty がゼロになった段階で、nWhite と nBlack の大小関係から勝敗を決定しています（nEmpty がゼロになるより前に勝敗の判定がつく場合について、具体的にプログラムする必要があるかもしれません）

print() メソッドは、画面に盤面の状態を表示しています

is_empty() メソッドは、引数で指定したスロットが空（Stone.EMPTY）であるか否かを判定しています

empty_list() メソッドは、現在の盤面において、空（Stone.EMPTY）のスロット番号の一覧をリストにして返します

can_put(te, color) メソッドは、引数 te で示されたスロット番号の場所に、第 2 引数の color で示された色の石を置けるかどうかを判定しています（具体的には、self.check_flip(Stone(te, color)) を呼び出して、色を反転できる石が何個かあるなら、そこには石を置けるという判断をしています）

check_flip(stone) メソッドでは、引数で受け取っている stone オブジェクトは、その石を置こうとしているスロットの位置番号、及びその石の色をプロパティとして持っていますから、盤面のその位置に指定された色の石を置いたときに、上下、左右、右斜め上、右斜め下、左斜め上、左斜め下の 8 つの方向で何個の石を反転できるかを数えています（最後に、反転できる石の総数を返しています）

flip_list(stone) メソッドでは、check_flip(stone) メソッドで数えた各方向での反転できる石の数に基づいて、反転する石のスロット番号のリストを作って返しています

put_stone(te, color) メソッドは、まずは、指定されたスロット位置に指定の石を置いた後に、反転する石のリストを flip_list(stone) メソッドで作って、そのリストを set_stones(flist,color) に渡して、実際に盤面の石を反転させるようにします

set_stones(flist,color) メソッドは、受け取ったリストを基に盤面のデータを更新し、石を反転させています

reset_stones() メソッドは、set_stones() メソッドで反転させた盤面上の石を元に戻します

puttable() メソッドは、can_put() メソッドが True だった場合の nKoma リストを取得します

puttable_list() メソッドは、puttbale() メソッドの戻り値から、盤面上で石を置ける場所のリストを取得します

ソースコード 7.4　Board class

```
1 from random import randint
2 from Stone import Stone
3
4 class Board:
5     PASS = -10
```

```
    GAME =  -1
    DRAW =   0
    HUMAN_WIN = BLACK_WIN = Stone.BLACK * 100
    MACHINE_WIN = WHITE_WIN = Stone.WHITE * 100
    UNITV = ((0,-1),(1,-1),(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1))

    def __init__(self, NW=4):
        self.nKoma = [0 for _ in range(9)]
        self.NW = NW
        self.NxN = NW * NW
        self.stones = [ Stone(i, Stone.EMPTY) for i in range(self.NxN) ]
        m = NW//2
        n = m - 1
        self.stones[NW*n+n].setColor( Stone.WHITE )
        self.stones[NW*n+m].setColor( Stone.BLACK )
        self.stones[NW*m+n].setColor( Stone.BLACK )
        self.stones[NW*m+m].setColor( Stone.WHITE )
        nrnd = randint(0,1)
        if nrnd==0:
            self.stones[NW * n + n].flipColor()
            self.stones[NW * n + m].flipColor()
            self.stones[NW * m + n].flipColor()
            self.stones[NW * m + m].flipColor()
        self.nBlack = self.nWhite = 2
        self.nEmpty = self.NxN - (self.nBlack + self.nWhite)
        self.state = Board.GAME

    def winner(self):
        self.nBlack = self.nWhite = self.nEmpty = 0
        for i in range(self.NxN):
            if self.stones[i].getColor() == Stone.HUMAN_ID:
                self.nBlack += 1
            elif self.stones[i].getColor() == Stone.MACHINE_ID:
                self.nWhite += 1
            else:
                self.nEmpty += 1
        if self.nBlack+self.nWhite == self.NxN:
            if self.nBlack < self.nWhite:
                return Board.MACHINE_WIN
            elif self.nBlack > self.nWhite:
                return Board.HUMAN_WIN
            else:
                return Board.DRAW
        return Board.GAME

    def print(self):
        print(f"WHITE:{self.nWhite}, BLACK:{self.nBlack}")
        print(' ',end=' ')
        for x in range(self.NW):
            print(f"{x}", end=' ')
        print("→x")
        for y in range(self.NW):
            print(f"{y:}", end=' ')
            for x in range(self.NW):
                print(f"{self.stones[ y*self.NW+x ].getFigure()}", end=' ')
            print()
        print("↓\ny")

    def check_flip(self, stone):
```

```
65          y = stone.getLocate()//self.NW
66          x = stone.getLocate()%self.NW
67          self.nKoma[8] = 0
68          if self.stones[y*self.NW+x].getColor() == Stone.EMPTY:
69              for i1 in range( len(self.nKoma)-1 ):   # 0,1,2,3,4,5,6,7
70                  m1, m2 = x, y
71                  self.nKoma[i1] = s = ct = 0
72                  while s<2:
73                      m1 += Board.UNITV[i1][0]
74                      m2 += Board.UNITV[i1][1]
75                      if (0<=m1<self.NW) and (0<=m2<self.NW):
76                          color = self.stones[m2*self.NW+m1].getColor()
77                          if color==Stone.EMPTY:
78                              s = 3
79                          elif color==stone.getColor():
80                              s=2 if s==1 else 3
81                          else:
82                              s=1
83                              ct += 1
84                      else:
85                          s=3
86                  if s==2:
87                      self.nKoma[8] += ct
88                      self.nKoma[i1] = ct
89          return self.nKoma[8]
90
91      def flip_list(self, stone):
92          self.check_flip(stone)
93          y = stone.getLocate()//self.NW
94          x = stone.getLocate()%self.NW
95          flipped = []
96          for i1 in range( len(self.nKoma)-1 ):   # 0,1,2,3,4,5,6,7
97              m1, m2 = x, y
98              for i2 in range( self.nKoma[i1] ):
99                  m1 += Board.UNITV[i1][0]
100                 m2 += Board.UNITV[i1][1]
101                 if (0 <= m1 < self.NW) and (0 <= m2 < self.NW):
102                     z = m2*self.NW+m1
103                     self.stones[z].setColor( stone.getColor() )
104                     flipped.append( z )
105         z = y*self.NW+x
106         self.stones[z].setColor( stone.getColor() )
107         flipped.append(z)
108         return flipped
109
110     def empty_list(self):
111         list = [v for v in range(self.NxN) if self.is_empty(v)]
112         return list
113
114     def is_empty(self, te):
115         return True if self.stones[te].getColor()==Stone.EMPTY else False
116
117     def can_put(self, te, color):
118         if 0<=te<self.NxN:
119             if 0<self.check_flip( Stone(te, color) ):
120                 return True
121         return False
122
123     def puttable(self, te, color):
```

```
124         if self.can_put(te, color):
125             return self.nKoma
126 
127     def puttable_list(self, color):
128         emptyl = self.empty_list()
129         komal = []
130         telist = []
131         for n in emptyl:
132             w = self.puttable(n, color)
133             if w:
134                 telist.append(n)
135                 komal.append(w)
136         return telist, komal
137 
138     def put_stone(self, te, color):
139         stone = Stone(te, color)
140         flist = self.flip_list( stone )
141         self.set_stones(flist, color)
142         self.stones[te] = stone
143         if color==Stone.BLACK:
144             self.nBlack = len(flist) + 1
145         else:
146             self.nWhite = len(flist) + 1
147         return flist
148 
149     def set_stones(self, flist, color):
150         for z in flist:
151             self.stones[z] = Stone(z, color)
152         self.stones[z] = Stone(z, Stone.EMPTY)
153 
154     def reset_stones(self, te, flist, color):
155         if color==Stone.WHITE:
156             c = Stone.BLACK
157         elif color==Stone.BLACK:
158             c = Stone.WHITE
159         self.stones[te] = Stone(te,c)
160         for z in flist:
161             self.stones[z] = Stone(z, c)
162         self.stones[z] = Stone(z, Stone.EMPTY)
163 
164     def debug_print(self):
165         for y in range(self.NW):
166             for x in range(self.NW):
167                 n = y*self.NW + x
168                 print(self.stones[n].color, end=' ')
169             print()
170         print('\n')
```

Human クラス

人の打つ石の手を決めているのは、このクラスのメソッド Te() です

プロンプトメッセージ（'Your turn. [yx] or "pass": '）を表示して、入力文字列を受け取ります（instr）

入力文字列（instr）の中に文字列'pass' が含まれていたなら、まず、石を置ける場所があったにもかかわらず、誤ってパスを指示したかもしれない可能性をチェック（puttable()）します

もし、石を置く場所がある局面なら、パスをしてはいけないので、プロンプトメッセージまで処理を戻

します（continue）

石を置く場所がなくてパスの指示をしたのなら、パスの指示を返します（return Board.PASS）

入力文字列にスロット番号として、行番号ｙと列番号ｘが [yx] として指定されたならば、ｙｘという 2 桁の数字文字を、それぞれ整数に変換（serial_num()）しています

ｙとｘが数字から整数値に変換できたなら、スロットの位置番号を計算（slot = cy * board.NW + cx）できます

計算したスロット番号に、その石を置けるかどうかをチェック（board.can_put(slot, self.color)）して、置けない場合は最初の文字列入力プロンプトまで処理を戻します

チェックした結果、石を置くことのできるスロット番号だったなら、反復（while True）を抜けて（break）、そのスロット番号を返します（return slot）

ソースコード 7.5　Human class

```
1  from Board import Board
2  from Stone import Stone
3
4  class Human:
5      def __init__(self, color=Stone.BLACK):
6          self.color = color
7
8      def Te(self, board):
9          def puttable():
10             elist =  board.empty_list()
11             for v in elist:
12                 if board.can_put(v, self.color):
13                     return True
14             return False
15
16         while True:
17             instr = input('Your turn. [yx] or "pass": ')
18             if 'pass' in instr:
19                 if puttable():
20                     continue
21                 return Board.PASS
22             if 1<len(instr) and self.numP( instr[0] ) and self.numP( instr[1] ):
23                 cy = self.serial_num( instr[0] )
24                 cx = self.serial_num( instr[1] )
25                 slot = cy * board.NW + cx
26                 if board.can_put(slot, self.color):
27                     break
28         print()
29         return slot
30
31     def numP(self, xy):
32         if '0' <= xy <= '9':
33             return True
34         return False
35
36     def serial_num(self, xy):
37         def fromA(xy):
38             return int(xy - 'a')
39
40         if self.numP(xy):
41             nxy = int( xy )
```

```
42          else:
43              nxy = fromA(xy)+10
44          return nxy
```

Machine クラス

コンピュータ（Machine）の打つ手を決めているのが、このクラスのメソッド、Te() です

①まず最初に、board.puttable_list() によって石を置ける場所の一覧を telist というリストとして取得します

②打つ手のリストの要素がない場合は、Board.PASS を返します

③打つ手のリストの要素が一つしかない場合は、迷うことなくその手を返します

④'random' 戦略が選ばれている場合は、telist の中から乱数で選んだ手を返します

⑤'maxflip' 戦略が選ばれている場合は、Board クラスの nKoma リストの一番最後の要素に、反転できる石の数の総数が入っているので、最も多く反転できる手を返します

⑥'minimax' 戦略が選ばれている場合は、MINIMAX 法による手が選ばれますが、MINIMAX 法は、勝敗がつくまで、最後まで対戦を試行するので、盤面のサイズが最小値（N=4）の場合でも、とても応答に時間がかかります。

⑦'alphabeta' 戦略が選ばれている場合は、MINIMAX 法を途中で打ち切る手立てを含んでいるので、多少高速な応答を期待できますが、、、、

ソースコード 7.6　Machine class

```
1  from random import randint
2  from Board import Board
3  from Stone import Stone
4  from Strategy import Strategy
5
6  class Machine( Strategy ):
7      def __init__(self, color=Stone.WHITE, strategy = 'random'):
8          self.color = color
9          self.strategy = strategy
10
11     def Te(self, board):
12         telist, komalist = board.puttable_list( self.color )
13         if not telist:
14             return Board.PASS
15         if len(telist)==1:
16             return telist[0]
17         if self.strategy == 'random':
18             n = randint( 0, len(telist)-1 )
19             return telist[n]
20         elif self.strategy == 'maxflip':
21             n = v = -1
22             for i, k in enumerate(komalist):
23                 if n < k[-1]:
24                     n = k[-1]
25                     v = telist[i]
26             return v
27         elif self.strategy == 'minimax':
28             print('thinking ...')
29             n = self.bestMove(board, telist)
30             print('Your turn.')
```

```
31              return n
32          elif self.strategy == 'alphabeta':
33              n = self.bestMoveAB(board, telist)
34              return n
```

7.1.2　Machine の戦略：Strategy クラス

このクラスは、Machine クラスのスーパークラスとして、Machine クラスに継承させて使うようにします

ここには、MINIMAX 法と Alpha-Beta 刈りの 2 つを記述しています

ソースコード 7.7　Strategy class

```
1  from Board import Board
2  from Stone import Stone
3
4  class Strategy:
5      INFINITY = 100000
6      def __init__(self):
7          pass
8
9      # MiniMax
10     def bestMove(self, board, telist):
11         bestEval = -Strategy.INFINITY
12         bestMove = -1
13         for n, v in enumerate(telist):
14             flist = board.put_stone(v, self.color)
15             eval = self.minimax(board, 0, False)
16             if bestEval<eval:
17                 bestEval = eval
18                 bestMove = v
19             board.reset_stones(v, flist, self.color)
20         return bestMove
21
22     def minimax(self, board, depth, isMaximizingPlayer):
23         def evaluate(depth):
24             if board.state == Board.MACHINE_WIN:
25                 board.state = Board.GAME
26                 return board.NxN-depth      #return NxN
27             elif board.state == Board.HUMAN_WIN:
28                 board.state = Board.GAME
29                 return depth-board.NxN      #return -NxN
30             else:
31                 self.state = Board.GAME
32                 return 0
33
34         board.winner()
35         if board.state != Board.GAME:
36             return evaluate(depth)
37
38         color = Stone.MACHINE_ID if isMaximizingPlayer else Stone.HUMAN_ID
39         telist, _ = board.puttable_list(color)
40         bestVal = -Strategy.INFINITY if isMaximizingPlayer else +Strategy.INFINITY
41         for n, v in enumerate(telist):
42             flist = board.put_stone(v, color)
43             value = self.minimax(board, depth + 1, not isMaximizingPlayer)
```

```
44              bestVal = max(value, bestVal) if isMaximizingPlayer else min(value,
                    bestVal)
45              board.reset_stones(v, flist, color)
46          return bestVal
47
48      # Alpha-Beta
49      def bestMoveAB(self, board, telist):
50          bestEval = -Strategy.INFINITY
51          bestMove = -1
52          for n, v in enumerate(telist):
53              flist = board.put_stone(v, self.color)
54              eval = self.minimaxab(board, 8, 0, False, -Strategy.INFINITY, +Strategy
                    .INFINITY)
55              if bestEval<eval:
56                  bestEval = eval
57                  bestMove = v
58              board.reset_stones(v, flist, self.color)
59          return bestMove
60
61      def minimaxab(self, board, node, depth, isMaximizingPlayer, alpha, beta):
62          def evaluate(depth):
63              if board.state == Board.MACHINE_WIN:
64                  board.state = Board.GAME
65                  return board.NxN-depth      #return NxN
66              elif board.state == Board.HUMAN_WIN:
67                  board.state = Board.GAME
68                  return depth-board.NxN      #return -NxN
69              else:
70                  self.state = Board.GAME
71                  return 0
72
73          board.winner()
74          if board.state != Board.GAME:
75              return evaluate(depth)
76          elif node==0:
77              n = board.nBlack - board.nWhite
78              return n if isMaximizingPlayer else -n
79
80          color = Stone.MACHINE_ID if isMaximizingPlayer else Stone.HUMAN_ID
81          telist, _ = board.puttable_list(color)
82          bestVal = -Strategy.INFINITY if isMaximizingPlayer else +Strategy.INFINITY
83          for n, v in enumerate(telist):
84              flist = board.put_stone(v, color)
85              value = self.minimaxab(board, node-1, depth+1, not isMaximizingPlayer,
                    alpha, beta)
86              board.reset_stones(v, flist, color)
87              if isMaximizingPlayer:
88                  bestVal = max(value, bestVal)
89                  alpha = max(alpha, bestVal)
90              else:
91                  bestVal = min(value, bestVal)
92                  beta = min(beta, bestVal)
93              if beta<=alpha:
94                  break
95          return bestVal
```

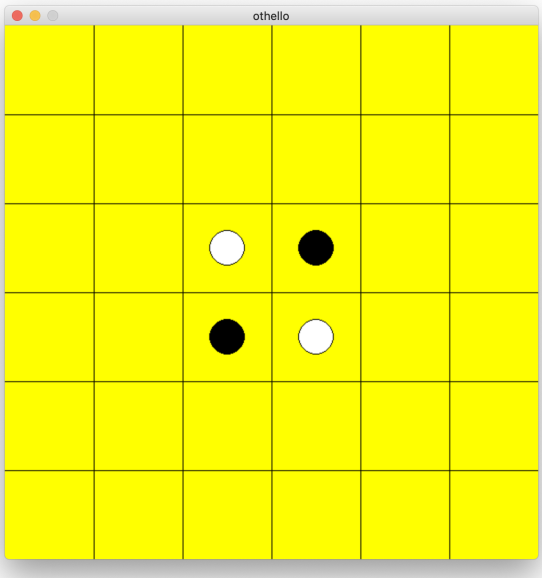
【演習】GUI 版に直してみましょう

7.2　GUI 版

人とコンピュータの対戦を GUI 化します

コンピュータが PASS の局面では、人の手番になるだけですが、人が PASS の判断をする局面では、何らかの方法でその旨を知らせてやる必要があります。真ん中の 4 つのスロットは空きになることはないので、その 4 つのどれかをクリックしたときには PASS を選択したことにします。ただし、人が PASS を選択してもどこかに石を置けるようでしたら、手番は交代せず、石を置けるスロットがクリックするのを待っています。

利口な戦略を選ぼうとすると、盤面のサイズとの兼ね合いで実用的なゲームにならないことがあります。Strategy クラスや Machine クラスを見て、そこで行っている処理を研究してみてください



main.py に変更なし

メインに変更はありません。Game クラスのコンストラクタでへの引数で盤面のサイズ（N=4 以上の偶数）、先手を'human', 'machine' から、戦略を'random', 'maxflip', 'minimax', 'alphabeta' から指定します

ソースコード 7.8　main

```
1 from Game import Game
2
3 if __name__ == '__main__':
4     sente = 'human'          # 'human', 'machine'
5     senryaku = 'maxflip'     # 'random', 'maxflip', 'minimax', 'alphabeta'
6     game = Game( N=6, turn=sente, strategy=senryaku )       # board size -> N x N
7     game.start()
```

Screen クラスの追加

ソースコード 7.9　Screen class

```
import pygame

class Screen:
    def __init__(self, NW=4, wh=(600, 600), bgcolor=(0, 0, 0)):
        self.NW = NW
        self.WIDTH = wh[0]
        self.HEIGHT = wh[1]
        self.COLOR = bgcolor
        self.SIZE = (self.WIDTH, self.HEIGHT)
        self.surface = pygame.display.set_mode(self.SIZE)
        self.SLOTW = self.WIDTH // NW
        self.SLOTH = self.HEIGHT // NW

    def fill(self):
        self.surface.fill(self.COLOR)
        BLACK = (0, 0, 0)
        for i in range(1, self.NW):
            startp = (0, self.SLOTH * i)
            endp = (self.WIDTH, self.SLOTH * i)
            pygame.draw.line(self.surface, BLACK, startp, endp)
            startp = (self.SLOTW * i, 0)
            endp = (self.SLOTW * i, self.HEIGHT)
            pygame.draw.line(self.surface, BLACK, startp, endp)

    def caption(self, str):
        pygame.display.set_caption(str)
```

Board クラスの書き換え

Board クラスは、Screen クラスを継承させることとし、Board クラスのコンストラクタの冒頭で、Screen クラスのコンストラクタを呼び出すようにします

盤面を surface に描画するメソッド、draw() を Board クラスの最後に追加します。draw() メソッドでは、self.stones のリストに持っている石の色に従って、白丸と黒丸を pygame.draw.circle() を使って描いています

ソースコード 7.10　Board class

```
from random import randint
import pygame
from Stone import Stone
from Screen import Screen

class Board( Screen ):    # Screenを継承するように変更
    PASS = -10
    GAME =  -1
    DRAW =   0
    HUMAN_WIN = BLACK_WIN = Stone.BLACK * 100
    MACHINE_WIN = WHITE_WIN = Stone.WHITE * 100
    UNITV = ((0,-1),(1,-1),(1,0),(1,1),(0,1),(-1,1),(-1,0),(-1,-1))

    def __init__(self, NW=4):
        # Screenのコンストラクタ呼びだしを追加
        super().__init__( NW=NW, wh=(600,600), bgcolor=(255,255,0) )
```

```
17          # コンストラクタの以下の部分に変更なし
18          self.nKoma = [0 for _ in range(9)]
19          self.NW = NW
20          self.NxN = NW * NW
21          self.stones = [ Stone(i, Stone.EMPTY) for i in range(self.NxN) ]
22          m = NW//2
23          n = m - 1
24          self.stones[NW*n+n].setColor( Stone.WHITE )
25          self.stones[NW*n+m].setColor( Stone.BLACK )
26          self.stones[NW*m+n].setColor( Stone.BLACK )
27          self.stones[NW*m+m].setColor( Stone.WHITE )
28          nrnd = randint(0,1)
29          if nrnd==0:
30              self.stones[NW * n + n].flipColor()
31              self.stones[NW * n + m].flipColor()
32              self.stones[NW * m + n].flipColor()
33              self.stones[NW * m + m].flipColor()
34          self.nBlack = self.nWhite = 2
35          self.nEmpty = self.NxN - (self.nBlack + self.nWhite)
36          self.state = Board.GAME
37
38      def winner(self):
39          ・
40          ・
41          変更なし
42          ・
43          ・
44
45      # 以下を、Boardクラスの最後に追加
46
47      def draw(self):
48          self.fill()
49          N = self.NW
50          RADIUS = int(self.SLOTW*0.2)
51          BLACK = (0,0,0)
52          for ypos in range(N):
53              yc = ypos * self.SLOTH + self.SLOTH // 2
54              for xpos in range(N):
55                  xc = xpos*self.SLOTW + self.SLOTW // 2
56                  i = ypos * N + xpos
57                  if self.stones[i].getColor() == Stone.WHITE:
58                      pygame.draw.circle(self.surface, (255,255,255), (xc,yc), RADIUS
                            )
59                      pygame.draw.circle(self.surface, BLACK, (xc, yc), RADIUS, 1)
60                  elif  self.stones[i].getColor() == Stone.BLACK:
61                      pygame.draw.circle(self.surface, BLACK, (xc, yc), RADIUS)
```

Game クラスの書き換え

start() メソッドの中で、Machine が石を置くときの処理を書き、key_event() メソッドの中で、MOUSEBUTTONUP のイベントを拾ったときに Human が盤面上に石を置く時の処理を書いています

window の caption への文字列の表示が効果的なものになるように、工夫してみましょう

ソースコード 7.11　Game class

```
1  import sys
```

```
2  import pygame
3  from pygame.locals import QUIT, MOUSEBUTTONUP
4  from Board import Board
5  from Human import Human
6  from Machine import Machine
7  from Stone import Stone
8
9  class Game():
10     def __init__(self, N=4, turn='human', strategy='random'):
11         pygame.init()
12         self.board = Board( NW=N )
13         self.board.caption('othello')
14         self.human = Human(color=Stone.BLACK)
15         self.machine = Machine(color=Stone.WHITE, strategy=strategy)
16         self.turn = False if turn == 'human' else True
17         self.clock = pygame.time.Clock()
18         self.FPS = 10
19
20         def key_event(self):
21             def fine():
22                 pygame.quit()
23                 sys.exit()
24             for event in pygame.event.get():
25                 if event.type == QUIT:
26                     fine()
27                 elif event.type == MOUSEBUTTONUP:
28                     if not self.turn:
29                         self.board.caption('Your turn')
30                         p = self.human.put_stone(self.board, event.pos)
31                         if p==True or p==Board.PASS:
32                             self.turn = not self.turn
33                             self.board.caption('Machine turn')
34                         else:
35                             pass        # DRAW
36
37     def judge(self):
38         result = False
39         self.board.winner()
40         if self.board.state == Board.MACHINE_WIN:
41             self.board.caption('Computer won the game!')
42         elif self.board.state == Board.HUMAN_WIN:
43             self.board.caption('You won!')
44         elif self.board.state == Board.DRAW:
45             self.board.caption('Draw!')
46         else:
47             #self.board.state == Board.GAME:
48             result = True
49         return result
50
51     def start(self):
52         while True:
53             self.key_event()
54             self.board.draw()
55             if self.turn:
56                 self.board.caption('Machine turn')
57                 p = self.machine.put_stone(self.board)
58                 if p==True or p==Board.PASS:
59                     self.turn = not self.turn
60                     self.board.caption('Your turn')
```

```
61              self.judge()
62              pygame.display.update()
63              self.clock.tick(self.FPS)
```

Machine クラスの書き換え

Machine クラスの最後に put_stone() メソッドを追加しますが、内容は、CUI の時の Te() メソッドを呼び出しているだけです

ソースコード 7.12　Machine class

```
1  from random import randint
2  from Board import Board
3  from Stone import Stone
4  from Strategy import Strategy
5
6  class Machine( Strategy ):
7      def __init__(self, color=Stone.WHITE, strategy = 'random'):
8          self.color = color
9          self.strategy = strategy
10
11     def Te(self, board):
12         ・
13         ・
14         変更なし
15         ・
16         ・
17
18     # 以下を、Machineクラスの最後に追加
19
20     def put_stone(self, board):
21         board.caption('thinking....')
22         te = self.Te(board)
23         if te!=board.PASS:
24             board.put_stone(te, self.color)
25             return True
26         return Board.PASS
```

Human クラスの書き換え

Human クラスでは、Game クラスの中から呼び出される、put_stone() メソッドを追加します

put_stone() メソッドでは、マウスボタンを離したイベントで受け取った screen 上の座標値を pos という名のタプルで受け取りますので、その座標がオセロ盤の何番のスロットをクリックしたのかを導出しています

また真ん中の 4 つのスロットは、PASS を指示するためのスロットでもあるので、その判定もしています

False を返す場合というのは、「もう一度呼びだし直してください」という意味になります

ソースコード 7.13　Human class

```
1  from Board import Board
2  from Stone import Stone
```

```
3
4   class Human:
5       def __init__(self, color=Stone.BLACK):
6           self.color = color
7
8       def Te(self, board):
9           ・
10          ・
11          変更なし
12          ・
13          ・
14
15      # 以下を、Humanクラスの最後に追加
16
17      def put_stone(self, board, pos=(-1,-1)):
18          def get_slot(xy):
19              x, y = xy[0], xy[1]
20              N = board.NW
21              for ypos in range(N):
22                  y0 = board.SLOTH * ypos
23                  y1 = board.SLOTH * (ypos + 1)
24                  for xpos in range(N):
25                      x0 = board.SLOTW * xpos
26                      x1 = board.SLOTW * (xpos + 1)
27                      if y0 < y < y1 and x0 < x < x1:
28                          return ypos * N + xpos
29              return -1
30          nw = board.NW
31          nwc = nw // 2
32          p1 = nw*(nwc-1) + nwc-1
33          p2 = p1 + 1
34          p3 = p1 + nw
35          p4 = p2 + nw
36          pass_slot = (p1, p2, p3, p4)
37          # 真ん中の4つのスロットはPASSの時に選択することにする
38          telist, komalist = board.puttable_list(self.color)
39          te = get_slot(pos)
40          if te in pass_slot:
41              if telist:
42                  return False
43              else:
44                  return Board.PASS
45          if te in telist:
46              board.put_stone(te, self.color)
47              return True
48          return False
```

Strategy クラスに変更なし

変更はないのですが、改善の余地はおおいに有りです

第 8 章

Appendix

8.1　MiniMax 法

8.1.1　Introduction

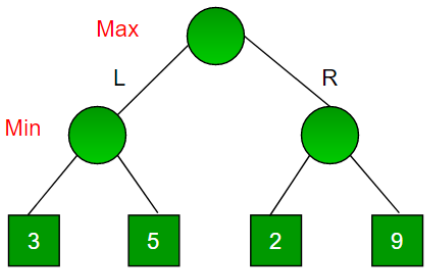
https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

MninMax は、バックトラック アルゴリズムの 1 つです。ゲーム プレーヤにとって最適の動きを見いだす（相手もまた最適の動きをすると仮定している）、そのためのゲーム理論や意思決定において使われています。このアルゴリズムは、2 人が交互にプレイするゲーム、例えば Tic-Tac-Toe, Backgammon, Mancala, Chess などで広く使われています。

Minimax において 2 人のプレーヤは、maximizer 及び minimizer と呼ばれています。maximizer は、できるだけ高いスコアを得ようとします。一方で、minimizer は逆に、できるだけ低いスコアを獲得しようとします。

各盤面の状態はそれぞれ、関係する値を持っています。与えられた状態において、もし minimizer が優勢であるなら、その後盤面のスコアは幾つかの正の値に移行しやすい。もし minimizer がその局面で優勢ならば、その値は負の値になりやすい。盤面の値は、ゲームのそれぞれのタイプに対してユニークな幾つかの経験によって計算されます。

例：4 つの最終状態と、最終状態に到達するための経路がルートから 4 つの葉に向かう完全な二分木を持つ、以下に示すようなゲームを考えてみましょう。あなたは maximizing player で、あなたが動く最初の機会を得ていると仮定しましょう。つまり、あなたはルートに居て、あなたと対戦している相手は次のレベルに居る状態です。あなたの相手もまた、最適な動きをするとき、あなたは maximizing player として、どちらの動きを採りますか？



これは、バックトラッキングアルゴリズムに基づくので、全ての可能な動きを試して、戻って、そして

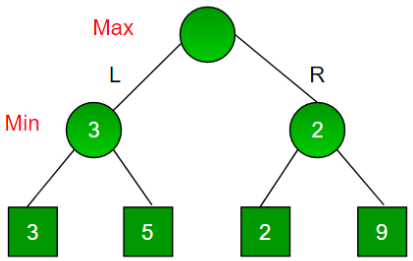
決断することになります。

（1）Maximizer が「左」へ動く場合：今度は minimizer の手番です。minimizer は 3 と 5 の間から選ぶ機会があります。minimizer になるためには、間違いなく両者の内での最小値を選ぶでしょう。その値は 3 です。

（2）Maximizer が「右」へ動く場合：今度は minimizer の手番です。minimizer は 2 と 9 の間から選ぶ機会があります。彼は 2 つの値の中の最小値として 2 を選ぶでしょう。

maximizer になるために、あなたは大きい方の値を選びます。その値は 3 です。従って、maximizer にとって最適な動きは「左」へ動くことであり、そのときの最適値は 3 です。

こうして、ゲームの木は以下のようになります。



上の木は、maximizer が左あるいは右に動く際の、2 つのあり得る値を示している。

注意：右側の枝にあたい 9 があるにもかかわらず、minimizer は決してその局面を選びません。私たちは常に、対戦相手は最適な手を取ると仮定しなければなりません。

以下は、同様の実装例です。

ソースコード 8.1　minimax000

```
# A simple Python3 program to find
# maximum score that
# maximizing player can get
import math

def minimax (curDepth, nodeIndex, maxTurn, scores, targetDepth):

    # base case : targetDepth reached
    if (curDepth == targetDepth):
        return scores[nodeIndex]

    if (maxTurn):
        return max(minimax(curDepth + 1, nodeIndex * 2, False, scores, targetDepth
            ),
                   minimax(curDepth + 1, nodeIndex * 2 + 1, False, scores,
                       targetDepth))

    else:
        return min(minimax(curDepth + 1, nodeIndex * 2, True, scores, targetDepth),
                   minimax(curDepth + 1, nodeIndex * 2 + 1, True, scores,
                       targetDepth))

# Driver code
scores = [3, 5, 2, 9, 12, 5, 23, 23]

```

```
treeDepth = math.log( len(scores), 2 )

print("The optimal value is : ", end = "")
print( minimax(0, 0, True, scores, treeDepth) )

# This code is contributed
# by rootshadow
```

Output:

The optimal value is: 12

この記事のアイデアは、簡単な実装を伴う Minimax への導入である。

上の例で、プレーヤーにとってただ 2 つだけの選択肢であった。一般的には、それ以上の選択肢になることがある。その場合には、私たちは全ての可能な動きに対しての再帰を実施して、最大／最小を見つける必要があります。例えば、Tic-Tax-Toe では、最初のプレーヤは 9 個の可能な動きを作ることができます。上の例で、スコア（＝ゲームの木の葉）は私たちに与えられています。典型的なゲームで、私たちはこれらの値を導き出す必要があります。私たちは、もうすぐ Minimax アルゴリズムによる Tic Tac Toe をカバーするようになるでしょう。

8.1.2　Introduction to Evaluation Function

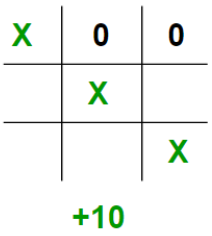
これまで見てきたように、各枝葉は関連する値を持っています。私たちはこの値を配列に納めることにした。しかし実際の世界では、Tic-Tac-Toe, Chess, Backgamon, などのゲームをするためのプログラムを作るとき、私たちは、盤面上の石の配置に依存した、盤面の値を計算する関数を実装する必要がある。この関数は、しばしば評価関数として知られている。しばしば発見的関数とも呼ばれる。

評価関数は、各ゲームのタイプ毎に異なります。本稿では、Tic-Tac-Toe のゲームの評価関数について議論します。評価関数の裏にある基本的なアイデアは、maximizer の手番なら、局面の値は高い値を与え、minimizer の手番なら、局面の値は低い値を与える。

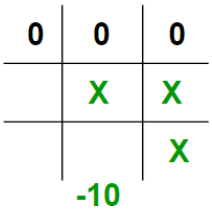
このシナリオについて、maximizer として X を、minimizer には O を考えることにしよう。

私たちの評価関数を作りましょう：

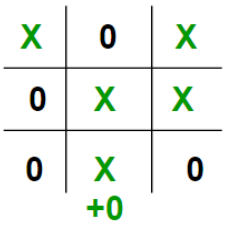
（1）もし X が勝ったなら、正の値＋ 10 を与える



（2）もし O が勝ったなら、負の値－ 10 を与える



(3) 誰も勝たなくて引き分けだったら、値＋ 0 を与える



私たちは、10 以外の正／負の値を選ぶこともできるが、簡単のために 10 を選んだ

ゲームのプレーヤと対戦者を、それぞれ小文字の'x' と'o' で表し、盤面の空きスロットをアンダースコア '_' であらわすことにする

また、盤面を 3 x 3 の 2 次元の文字マトリクス、char board[3][3]; のようにして表すとすれば、プレーヤのどちらかが 3 個一列に並べたかどうかをチェックするために、私たちは各行、各列、そして対角をチェックしなければならない

ソースコード 8.2　minimax001

```
# Python3 program to compute evaluation
# function for Tic Tac Toe Game.

# Returns a value based on who is winning
# b[3][3] is the Tic-Tac-Toe board
def evaluate(b):

    # Checking for Rows for X or O victory.
    for row in range(0, 3):

        if b[row][0] == b[row][1] and b[row][1] == b[row][2]:

            if b[row][0] == 'x':
                return 10
            elif b[row][0] == 'o':
                return -10

    # Checking for Columns for X or O victory.
    for col in range(0, 3):

        if b[0][col] == b[1][col] and b[1][col] == b[2][col]:

            if b[0][col]=='x':
                return 10
            elif b[0][col] == 'o':
                return -10

```

```
    # Checking for Diagonals for X or O victory.
    if b[0][0] == b[1][1] and b[1][1] == b[2][2]:

        if b[0][0] == 'x':
            return 10
        elif b[0][0] == 'o':
            return -10

    if b[0][2] == b[1][1] and b[1][1] == b[2][0]:

        if b[0][2] == 'x':
            return 10
        elif b[0][2] == 'o':
            return -10

    # Else if none of them have won then return 0
    return 0

# Driver code
if __name__ == "__main__":

    board = [['x', '_', 'o'],
             ['_', 'x', 'o'],
             ['_', '_', 'x']]

    value = evaluate(board)
    print("The value of this board is", value)

# This code is contributed by Rituraj Jain
```

Output:

The value of this board is 10

本稿のアイデアは、Tic-Tac-Toe のゲームの簡単な評価関数の書き方を理解することです。

次の原稿では、MiniMax アルゴリズムの中に評価関数を一緒にする方法について記述することにします

8.1.3　Tic-Tac-Toe AI – Finding optimal move

完璧なゲームをする適切な Tic-Tac-Toe AI を書くために、これまで学んだ MiniMax と評価関数を一緒にすることにしましょう

この AI は全ての可能なシナリオを考え、最も適切な動きをします

Finding the Best Move :

私たちは、findBestMove() と呼ばれる新しい関数を導入しましょう

この関数は、minimax() 関数を使って全てのあり得る動きを評価します。そして、maximizer が打てる最善の手を返します

擬似コードで書くと以下の通り

```
function findBestMove(board):
    bestMove = NULL
```

```
    for each move in board :
        if current move is better than bestMove
            bestMove = current move
    return bestMove
```

minimax :

現在の動きが、最善手よりも良い手かどうかをチェックするために、minimax() 関数は、ゲームの進行するあらゆる可能なやり方を考え、（相手もまた最善手を採ってくるとの仮定の下）最善手を返します。minimax() 関数における maximizer 及び minimizer のためのコードは、findBestMove() 関数と似ています。唯一の違いは、動きを返す代わりに、値を返すという点です。ここに、擬似コードを示します。

```
function minimax(board, depth, isMaximizingPlayer):

    if current board state is a terminal state :
        return value of the board

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each move in board :
            value = minimax(board, depth+1, false)
            bestVal = max( bestVal, value)
        return bestVal

    else :
        bestVal = +INFINITY
        for each move in board :
            value = minimax(board, depth+1, true)
            bestVal = min( bestVal, value)
        return bestVal
```

Checking for GameOver state :

ゲームが終了しているかどうかをチェックするため、あるいはもう打つ場所が残っていないかどうかに気づくため、isMovesLeft() 関数を使います。

これは、駒の動きがあるかないかについて、真あるいは偽をそれぞれ返すという簡単な関数です擬似コードは以下の通り

```
function isMovesLeft(board):
    for each cell in board:
        if current cell is empty:
            return true
    return false
```

Making our AI smarter :

最終ステップの 1 つは、私たちの AI をもうすこし賢くすることです。たとえ以下の AI が完璧なゲームを行ったとしても、ゆっくりと勝つか、あるいは速く負けるような結果になる手を選ぶかもしれない。例を使って説明していきます。与えられた盤面の状況から、X にとってゲームに勝つための 2 つの方法があると仮定します。

Move A :　X は 2 手で勝つ
Move B :　X は 4 手で勝つ

私たちの評価関数は、A の手も B の手も両方とも＋ 10 を値として返します。たとえ、A の手が速い勝ちを確定しているからいい手であるにもかかわらず、われわれの AI はしばしば B の手を選択する。この問題を克服するために、私たちは、評価値から探索の深さを引き算します。このことは、勝ちの場合に、それは最小手での勝利を選び、負ける場合には、できるだけ多くの手を打ってゲームを長引かせようとする。こうして、新しい評価値は次のようになる

Move A　は次の値にする $+10 - 2 = 8$
Move B　は次の値にする $+10 - 4 = 6$

A の手は B の手に比べて高いスコアを持っているので、私たちの AI は手 A を選ぶ。同様のことが minimizer に対しても適用されなければならない。探索深さを引き算する代わりに、私たちは、minimizer が常に、できるだけ負の値を得ようとして、その深さを加算します。評価関数の中でも外でも、どちらでも探索深さを引き算することができる。どこでも大丈夫。私は、関数の外で引き算をすることを選びました。擬似コードは次のように実装できる

```
if maximizer has won:
    return WIN_SCORE -  depth

else if minimizer has won:
    return LOOSE_SCORE + depth
```

ソースコード 8.3　minimax003

```
1  // Java program to find the
2  // next optimal move for a player
3  class GFG
4  {
5  static class Move
6  {
7      int row, col;
8  };
9
10 static char player = 'x', opponent = 'o';
11
12 // This function returns true if there are moves
13 // remaining on the board. It returns false if
14 // there are no moves left to play.
15 static Boolean isMovesLeft(char board[][])
```

```
{
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            if (board[i][j] == '_')
                return true;
    return false;
}

// This is the evaluation function as discussed
// in the previous article ( http://goo.gl/sJgv68 )
static int evaluate(char b[][])
{
    // Checking for Rows for X or O victory.
    for (int row = 0; row < 3; row++)
    {
        if (b[row][0] == b[row][1] &&
            b[row][1] == b[row][2])
        {
            if (b[row][0] == player)
                return +10;
            else if (b[row][0] == opponent)
                return -10;
        }
    }

    // Checking for Columns for X or O victory.
    for (int col = 0; col < 3; col++)
    {
        if (b[0][col] == b[1][col] &&
            b[1][col] == b[2][col])
        {
            if (b[0][col] == player)
                return +10;

            else if (b[0][col] == opponent)
                return -10;
        }
    }

    // Checking for Diagonals for X or O victory.
    if (b[0][0] == b[1][1] && b[1][1] == b[2][2])
    {
        if (b[0][0] == player)
            return +10;
        else if (b[0][0] == opponent)
            return -10;
    }

    if (b[0][2] == b[1][1] && b[1][1] == b[2][0])
    {
        if (b[0][2] == player)
            return +10;
        else if (b[0][2] == opponent)
            return -10;
    }

    // Else if none of them have won then return 0
    return 0;
}
```

```
75
76  // This is the minimax function. It considers all
77  // the possible ways the game can go and returns
78  // the value of the board
79  static int minimax(char board[][],
80                     int depth, Boolean isMax)
81  {
82      int score = evaluate(board);
83
84      // If Maximizer has won the game
85      // return his/her evaluated score
86      if (score == 10)
87          return score;
88
89      // If Minimizer has won the game
90      // return his/her evaluated score
91      if (score == -10)
92          return score;
93
94      // If there are no more moves and
95      // no winner then it is a tie
96      if (isMovesLeft(board) == false)
97          return 0;
98
99      // If this maximizers move
100     if (isMax)
101     {
102         int best = -1000;
103
104         // Traverse all cells
105         for (int i = 0; i < 3; i++)
106         {
107             for (int j = 0; j < 3; j++)
108             {
109                 // Check if cell is empty
110                 if (board[i][j]=='_')
111                 {
112                     // Make the move
113                     board[i][j] = player;
114
115                     // Call minimax recursively and choose
116                     // the maximum value
117                     best = Math.max(best, minimax(board,
118                                     depth + 1, !isMax));
119
120                     // Undo the move
121                     board[i][j] = '_';
122                 }
123             }
124         }
125         return best;
126     }
127
128     // If this minimizers move
129     else
130     {
131         int best = 1000;
132
133         // Traverse all cells
```

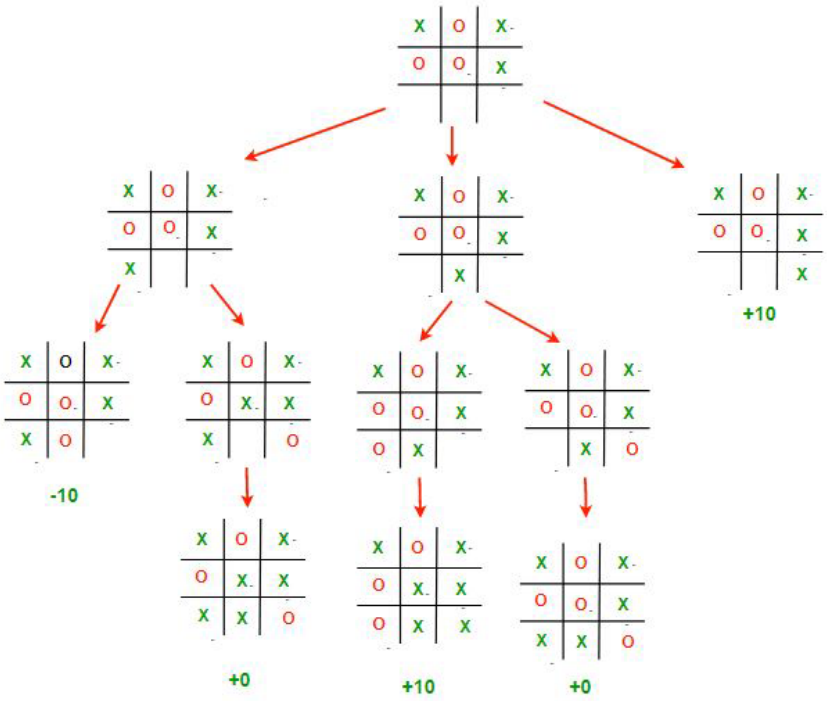
```
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                // Check if cell is empty
                if (board[i][j] == '_')
                {
                    // Make the move
                    board[i][j] = opponent;

                    // Call minimax recursively and choose
                    // the minimum value
                    best = Math.min(best, minimax(board,
                                    depth + 1, !isMax));

                    // Undo the move
                    board[i][j] = '_';
                }
            }
        }
        return best;
    }
}

// This will return the best possible
// move for the player
static Move findBestMove(char board[][])
{
    int bestVal = -1000;
    Move bestMove = new Move();
    bestMove.row = -1;
    bestMove.col = -1;

    // Traverse all cells, evaluate minimax function
    // for all empty cells. And return the cell
    // with optimal value.
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            // Check if cell is empty
            if (board[i][j] == '_')
            {
                // Make the move
                board[i][j] = player;

                // compute evaluation function for this
                // move.
                int moveVal = minimax(board, 0, false);

                // Undo the move
                board[i][j] = '_';

                // If the value of the current move is
                // more than the best value, then update
                // best/
                if (moveVal > bestVal)
                {
                    bestMove.row = i;
```

```
                    bestMove.col = j;
                    bestVal = moveVal;
                }
            }
        }
    }

    System.out.printf("The value of the best Move " +
                           "is : %d\n\n", bestVal);

    return bestMove;
}

// Driver code
public static void main(String[] args)
{
    char board[][] = {{ 'x', 'o', 'x' },
                      { 'o', 'o', 'x' },
                      { '_', '_', '_' }};

    Move bestMove = findBestMove(board);

    System.out.printf("The Optimal Move is :\n");
    System.out.printf("ROW: %d COL: %d\n\n",
              bestMove.row, bestMove.col );
}

}

// This code is contributed by PrinciRaj1992
```

Output:

The value of the best Move is : 10

The Optimal Move is : ROW: 2 COL: 2



図は、ルートにある盤面の局面から、ゲームが取り得る可能な全ての手を描画しています。これは、しばしばゲームの木と呼ばれています。上の例で、3 つの可能なシナリオは以下の通りです

Left Move :　もし X が [2,0] を選んだら．相手の O は [2,1] を選んでゲームに勝つでしょう．この手の評価値 -10

Middle Move :　もし X が [2,1] を選んだら．相手の O は [2,2] を選んでゲームは引き分けるでしょう．この手の評価値は 0

Right Move :　もし X が [2,2] を選んだら．X はゲームに勝つでしょう．この手の評価値は +10

X が 2 つめの手でゲームに勝つ可能性があったとしても、相手の O は決してその偶然を起こさないし、その代わり引き分けをえらぶでしょう。従って、X にとっての最善手は [2,2] となり、この手は X に勝利を保証するでしょう。

8.1.4　Alpha-Beta Pruning

「Alpha-Beta 刈り」は実際に新しいアルゴリズムではありません。どちらかと言えば、minimax アルゴリズムに対する最適化技術なのです。このアルゴリズムは、大きな要因でコンピュータの計算時間を減らします。これは、私たちにより速い探索を許し、ゲームの木のより深いレベルまで入り込むことを許します。それは、最善手が既に存在しているから、探索される必要のないゲームの木の枝を刈ります。これが「Alpha-Beta 刈り」と呼ばれるのは、minimax 関数に渡す 2 つの特別なパラメータがあって、それに alpha と beta と名前が付けられているからです。

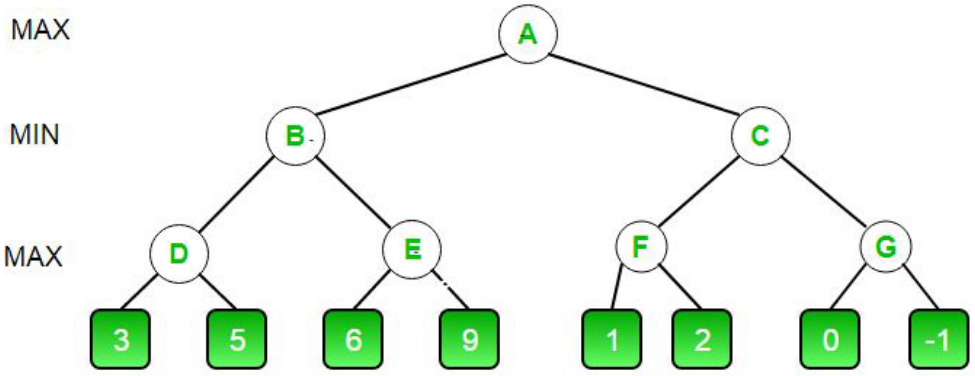
パラメータ alpha と beta を決定しましょう。Alpha は、現段階で maximizer が保証できるレベルあるいはそれ以上の最善手の評価値 Beta は、現段階で minimizer が保証できるレベルあるいはそれ以上の最善手の評価値

擬似コード：

```
function minimax(node, depth, isMaximizingPlayer, alpha, beta):
```

```
    if node is a leaf node :
        return value of the node

    if isMaximizingPlayer :
        bestVal = -INFINITY
        for each child node :
            value = minimax(node, depth+1, false, alpha, beta)
            bestVal = max( bestVal, value)
            alpha = max( alpha, bestVal)
            if beta <= alpha:
                break
        return bestVal

    else :
        bestVal = +INFINITY
        for each child node :
            value = minimax(node, depth+1, true, alpha, beta)
            bestVal = min( bestVal, value)
            beta = min( beta, bestVal)
            if beta <= alpha:
                break
        return bestVal

// Calling the function for the first time.
minimax(0, 0, true, -INFINITY, +INFINITY)
```

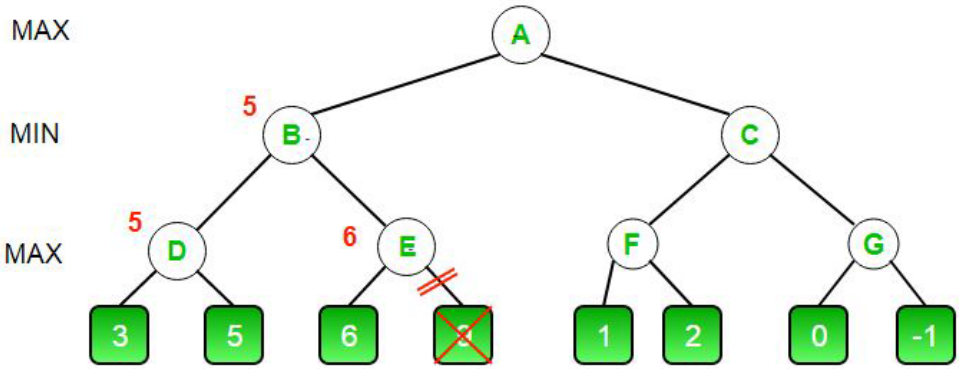
例を使って、上のアルゴリズムを見ていきます



- 最初は A からスタートします。ここでの alpha の値は-INFINITY そして beta の値は +INFINITY です. これらの値はゲームの木の後続の節に伝えられます。A において、maximizer は B と C の中の最大値を選ばなければなりません。従って A は B を最初に呼び出します

- B において、minimizer は D と E の中の最小値を選ばなければなりません。従って D を最初に呼び出します
- D において、左の枝葉を見ます。この節は値として 3 を返します。今、D における alpha の値は max(-INF, 3) で、その値は 3 です。
- 右の節を見る必要があるかないかを決めるために、*beta* <= *alpha* をチェックします。今、beta = +INF で alpha = 3 ですから、このチェック条件は False です。従って、探索を継続しなければなりません。
- 今、D はその右の節（値 5 を返す） を見ます.D において、alpha = max(3, 5) これは 5. こうして D における値は 5 です
- D は値 5 を B に返します. B において, beta = min(+INF, 5) これは 5. minimizer は値 5 あるいはそれ以下の値を保証します. 5 より小さい値が得られるかどうかを見るために、B はここで E を呼びます.
- E において、alpha と beta の値は、それぞれ -INF と +INF ではなくて、そのかわり -INF と 5 になります, なぜなら、beta の値は B において変わったからです そしてそれは、B が E に伝えた値です
- 今 E は左の節を見て、その値は 6 です. E において, alpha = max(-INF, 6) これは 6. 従って条件は True になります. beta は 5 で alpha は 6 です. 従って *beta* <= *alpha* は True です. 従って breaks して、 E は B に 6 を返します
- E の右の節の値が幾つかということを、いかに気にしなかったか注しましょう. 値は +INF あるいは -INF になる可能性がある, それはまだ問題かもしれない、私たちは、決して見る必要はない、なぜなら minimizer は、5 あるいはそれ以下を保証されるからです。従って maximizer が 6 を見るやいなや、彼は minimizer が決してこの手を採らないことを知ることになる。なぜならば、彼は左側の B で 5 を得ることができるからです。この方法は私たちが 9 を見る必要がないこと、そして従ってコンピュータの計算時間を節約できる
- E は B に値 6 を返します. B において, beta = min(5, 6) これは 5.　 B の節の値もまた 5 です

これまで、こうやってゲームの木を見ます。決して計算されなかった値 9 は取り消されます

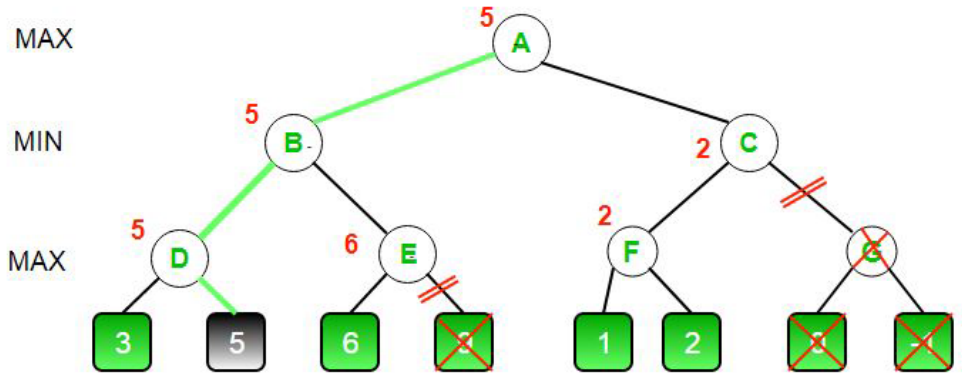


- B は A に 5 を返します. A において, alpha = max(-INF, 5) これは 5. 今 maximizer は 5 あるいはそれ以上の値を保証します. A は今 5 以上の値を得られるかどうかを見るために C を呼びます.
- C において, alpha = 5 そして beta = +INF　 です. C は F を呼びます
- F において, alpha = 5 そして beta = +INF　 です. F は値 1 の左の節を見ます. alpha = max(

5, 1) これはまだ 5.

- F は値 2 の右の節を見ます. こうしてこの節のベストな値は 2 です. Alpha はまだ 5 のままです
- F は C に値 2 を返します. C において, beta = min(+INF, 2). 条件 *beta <= alpha* True になります、beta = 2 で alpha = 5 だからです. そして、it breaks して、 G 以下の枝全体を計算する必要さえありません.
- この枝切りの背後の直感は、C において、minimizer は 2 あるいはそれ以下の値を保証したことですしかし、maximizer は既に、もし彼が B を選んだら値 5 を保証しました. そうして、なぜ maximizer がこれまで C を選んで、そして 2 以下の値を得ることになる可能性があるのか？再び、それらの最後の 2 つの値が何なのかを気にしなかったことを見ることができます。私たちは、以下の枝全体をスキップすることによって、コンピュータの計算を節約できます
- C は今 A に値 2 を返します. 従って A における最善手は max(5, 2) これは 5 です.
- こうして、最適値として maximizer は値 5 を得られます

このようにして、私たちの最終的なゲームの木はこのようになります G が取り消されることが分かります。決して計算されません



ソースコード 8.4　minimax004

```
# Python3 program to demonstrate
# working of Alpha-Beta Pruning

# Initial values of Aplha and Beta
MAX, MIN = 1000, -1000

# Returns optimal value for current player
#(Initially called for root and maximizer)
def minimax(depth, nodeIndex, maximizingPlayer,
            values, alpha, beta):

    # Terminating condition. i.e
    # leaf node is reached
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:

        best = MIN

        # Recur for left and right children
```

```
        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex * 2 + i,
                          False, values, alpha, beta)
            best = max(best, val)
            alpha = max(alpha, best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break

        return best

    else:
        best = MAX

        # Recur for left and
        # right children
        for i in range(0, 2):

            val = minimax(depth + 1, nodeIndex * 2 + i,
                          True, values, alpha, beta)
            best = min(best, val)
            beta = min(beta, best)

            # Alpha Beta Pruning
            if beta <= alpha:
                break

        return best

# Driver Code
if __name__ == "__main__":

    values = [3, 5, 6, 9, 1, 2, 0, -1]
    print("The optimal value is :", minimax(0, 0, True, values, MIN, MAX))

# This code is contributed by Rituraj Jain
```

Output:

The optimal value is : 5

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] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/
[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」