プログラミング教室のテクノロ

Pythonの道

本格的なゲーム開発②(中編)

もくじ

・パズルゲームをつくる（中編）

　複数のアルゴリズムを組み込む方法を学ぶ

マス上のデータを管理する

格子状に並ぶネコ（ブロック）をリストで管理するプログラムを作る。格子は横8マス、縦10マスとし、マスには何にも入ってないか、6種類のネコの内、いずれかが入る。このようなゲーム画面は二次元リストを使って管理する。

neko[y][x]の値	0	1	2	3	4	5	6	7
	何も表示しない	neko1.png 	neko2.png 	neko3.png 	neko4.png 	neko5.png 	neko6.png 	neko_niku.png 

二次元リスト

```
neko=[
[1,0,0,0,0,0,3,7],
[0,2,0,0,4,0,5,7],
[0,0,3,0,0,0,0,0]
]
```

ネコブロックを配置する関数

```
def ネコ配置():
    for y in range(10):
        for x in range(8):
            if neko[y][x] > 0:
                cvs.create_image(x,y,image=img_neko[neko[y][x]])
```

② ①

(0,0),(0,1),(0,2),……
(1,0),(1,1),(1,2),……
(2,0),(2,1),(2,2),……
.
.

ネコブロックの画像を保存するリスト

```
img_neko=[
    None,                                          ⇒0
    tkinter.PhotoImage(file="neko1.png"),          ⇒1
    tkinter.PhotoImage(file="neko2.png"),          ⇒2
    tkinter.PhotoImage(file="neko3.png"),          ⇒3
    tkinter.PhotoImage(file="neko4.png"),          ⇒4
    tkinter.PhotoImage(file="neko5.png"),          ⇒5
    tkinter.PhotoImage(file="neko6.png"),          ⇒6
    tkinter.PhotoImage(file="neko_niku.png")       ⇒7
]
```

③

二次元リストを理解する（復習）

表計算で扱うような行と列の2次元の表は二次元リストを使用することで扱うことができる。

国名	感染者	死者	回復者
米　国	8,134,334	219,355	3,604,780
日　本	93,931	1,688	86,144
中　国	85,672	4,634	80,786
台　湾	535	7	491

これを2次元リストで表すとこうなる。

```
感染者数 = [
    ["国名", "感染者", "死者", "回復者"],
    ["米国", 8134334, 219355, 3604780],
    ["日本", 93931, 1688, 86144],
    ["中国", 85672, 4634, 80786],
    ["台湾", 535, 7, 491],
]
```

二次元リストから値を取り出す方法①

```
print(感染者数[1])
```

```
['米国', 8134334, 219355, 3604780]
>>>
```

二次元リストから値を取り出す方法②

```
print(感染者数[1][0])
```

```
米国
>>>
```

二次元リストから値を取り出す方法③

```
for i in range(4):
    print(感染者数[i])
```

```
['国名', '感染者', '死者', '回復者']
['米国', 8134334, 219355, 3604780]
['日本', 93931, 1688, 86144]
['中国', 85672, 4634, 80786]
>>>
```

二次元リストから値を取り出す方法④

```
for i in range(5):
    for j in range(4):
        print(感染者数[i][j])
```

```
国名
感染者
死者
回復者
米国
8134334
219355
3604780
日本
93931
1688
86144
中国
85672
4634
80786
台湾
535
7
491
>>>
```

二次元リストを理解する（復習）

二次元リストに数字を入れ、数字に応じた図形を表示させてみよう。

```
a = [
    [0,1,2,3],
    [0,0,1,1],
    [1,1,1,2],
]
```

二次元リストを
数字で作成

```
for i in range(3):
    for j in range(4):

        if a[i][j]==0:
            print("■")
        elif a[i][j]==1:
            print("〇")
        elif a[i][j]==2:
            print("▲")
        else:
            print("× ")
```

```
a = [
    [0,1,2,3],
    [0,0,1,1],
    [1,1,1,2],
]
```

```
0,0
0,1
0,2
0,3
1,0
1,1
1,2
1,3
2,0
2,1
2,2
2,3
```

```
⇒ 0
⇒ 1
⇒ 2
⇒ 3
⇒ 0
⇒ 0
⇒ 1
⇒ 1
⇒ 1
⇒ 1
⇒ 1
⇒ 2
```

■
〇
▲
×
■
■
〇〇〇
〇〇〇
〇〇〇
▲
>>>

```
if a[i][j]==0:
    print("■")
elif a[i][j]==1:
    print("〇")
elif a[i][j]==2:
    print("▲")
else:
    print("× ")
```

図形を画像に変えると
パズルゲームになる。

マス上のデータを管理する

格子状に並ぶネコブロックをリストで管理するプログラムを作成する。

```
import tkinter  ・・・・tkinterモジュールの呼出し

neko=[              ・・・・マス目を管理する二次元リスト
[1,0,0,0,0,0,7,7],
[0,2,0,0,0,0,7,7],
[0,0,3,0,0,0,0,0],
[0,0,0,4,0,0,0,0],
[0,0,0,0,5,0,0,0],
[0,0,0,0,0,6,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,1,2,3,4,5,6]
]

def draw_neko():  ・・・・ネコを表示する関数
    for y in range(10):    ・・・繰り返し yは0から9まで1ずつ増える
        for x in range(8): ・・・繰り返し xは0から9まで1ずつ増える
            cvs.create_image(x*72+60,y*72+60,image=img_neko[neko[y][x]])
            ・・・img_nekoに指定した画像を表示する
```

次のページに続く

マス上のデータを管理する

```
root=tkinter.Tk()  ・・・ ウィンドウのオブジェクトを作成
root.title("二次元リストでマスを管理する")・・・ ウィンドウのタイトルを表示する
root.resizable(False,False) ・・・ウィンドウサイズを変更できないようにする
cvs=tkinter.Canvas(root,width=912,height=768) ・・・キャンバスの部品を作る
cvs.pack()  ・・・キャンバスを配置する

bg = tkinter.PhotoImage(file="neko_bg.png") ・・・背景画像の読み込み
img_neko=[                                  ・・・リストで複数のネコの画像を管理
    None,                                   Noneは何も表示しないという意味
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png")
]
cvs.create_image(456,384,image=bg)  ・・・キャンバス上に背景を描く
draw_neko()      ・・・ネコを表示する関数を呼び出す
root.mainloop()  ・・・ウィンドウを表示する
```

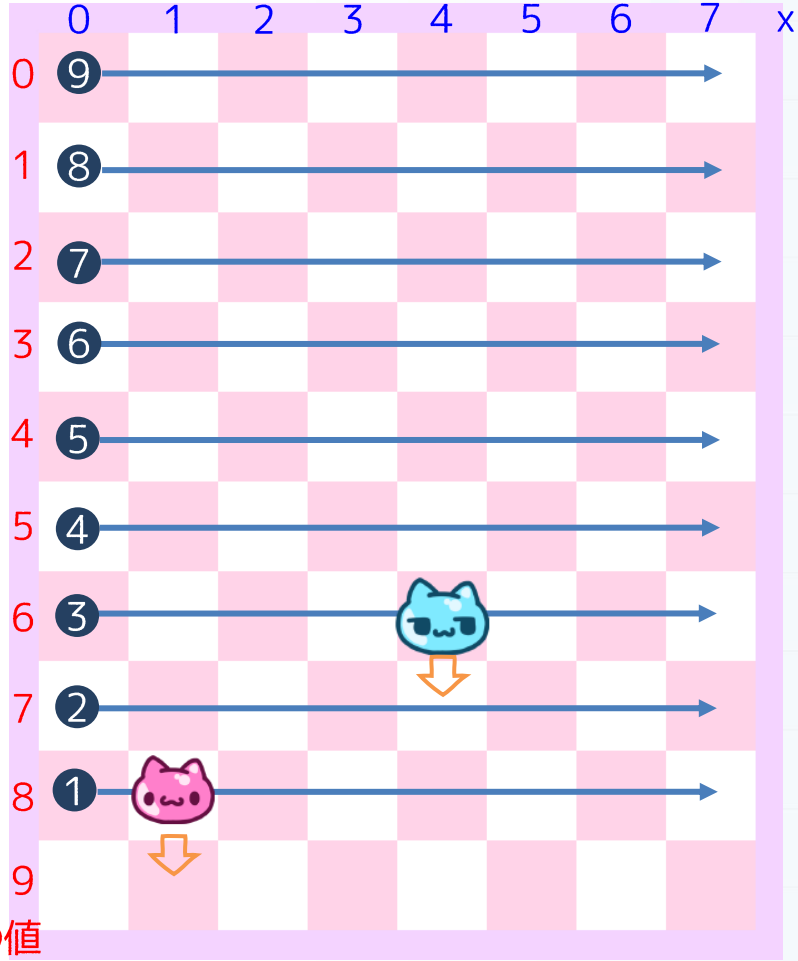
img_neko	0	1	2	3	4	5	6	7
	None	 neko1.png	 neko2.png	 neko3.png	 neko4.png	 neko5.png	 neko6.png	 neko_niku.png

マス上のデータを管理する

「Run Module」またはF5キーを押してプログラムを実行する。

ブロックを落下させるアルゴリズム

ブロックを落下させるには、ネコが存在するマスの１つ下のマスが空白の時、ネコをそこに移動させれば一段落下させることができる。一段ずつ落下させるには、下の段から上の段に向かって調べていく必要がある。




```
def drop_neko():
    for y in range(8,-1,-1):
        for x in range(8):
            if neko[y][x] != 0 and neko[y+1][x]==0:
                neko[y+1][x] = neko[y][x]
                neko[y][x] = 0
```

開始 終了

①の段から調べ始める。二重ループのforでyの値が8のとき、xの値は0→1→2→3→4→5→6→7と変化しながら横に調べていき、落とすべきネコがあれば一段落下させる。次にyの値が7になり②の段のところを調べていく。こうしてyの値が0、xの値が7になるまで二重ループの繰り返しが行われる。

ブロックを落下させるアルゴリズム

ネコが存在するマスの1つ下のマスが空白のとき、ネコをそこに移動させてネコが落下するプログラムを作る。

```
import tkinter ・・・・tkinterモジュールの呼出し

neko=[          ・・・・マス目を管理する二次元リスト
[1,0,0,0,0,0,7,7],
[0,2,0,0,0,0,7,7],
[0,0,3,0,0,0,0,0],
[0,0,0,4,0,0,0,0],
[0,0,0,0,5,0,0,0],
[0,0,0,0,0,6,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,1,2,3,4,5,6]
]

def draw_neko():  ・・・・ネコを表示する関数
    for y in range(10): ・・・繰り返し yは0から9まで1ずつ増える
        for x in range(8):  ・・・繰り返し xは0から7まで1ずつ増える
            cvs.create_image(x*72+60,y*72+60,image=img_neko[neko[y][x]],tag="NEKO")
                    ・・・img_nekoに指定した画像を表示する
```

次のページに続く

ブロックを落下させるアルゴリズム

```
def drop_neko():・・・・ネコを落下させる関数
    for y in range(8,-1,-1):・・・・繰り返し yは8から0まで1ずつ減る
        for x in range(8):  ・・・・繰り返し xは0から7まで1ずつ増える
            if neko[y][x] != 0 and neko[y+1][x]==0:  ・・・・ネコのあるマスの下が空白なら空白にネコを入れ、元のマスは空白にする
                neko[y+1][x] = neko[y][x]
                neko[y][x] = 0

def game_main():・・・・メインの処理を行う関数
    drop_neko()  ・・・・ネコを落下させる関数を呼び出す
    cvs.delete("NEKO")・・・・ネコの画像を削除
    draw_neko()       ・・・・ネコを表示
    root.after(100,game_main)・・・・0.1秒後に再びメインの処理を実行

root=tkinter.Tk()  ・・・・ウィンドウのオブジェクトを作る
root.title("ネコを落下させる")・・・・タイトルを指定する
root.resizable(False,False)    ・・・・ウィンドウのサイズを変更できないようにする
cvs=tkinter.Canvas(root,width=912,height=768)・・・キャンバスの部品を作る
cvs.pack()・・・キャンバスを配置する
```

次のページに続く

ブロックを落下させるアルゴリズム

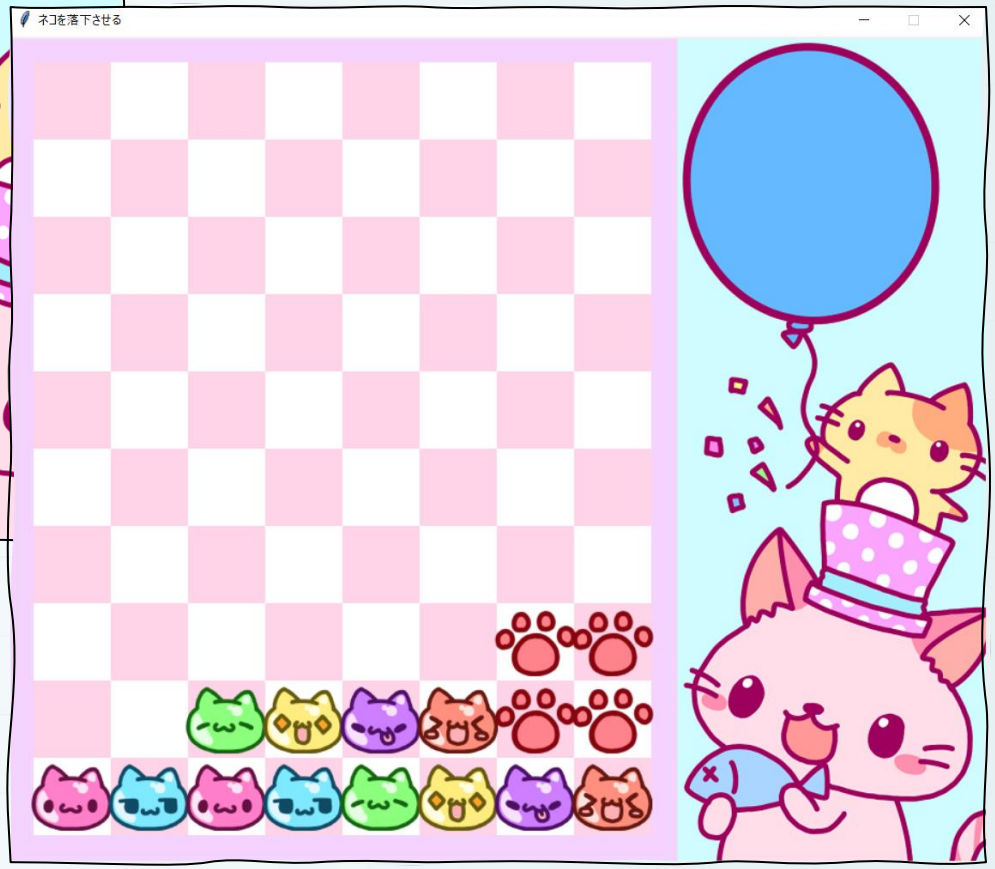
```
bg = tkinter.PhotoImage(file="neko_bg.png")  ・・・背景画像の読み込み
img_neko=[                                   ・・・リストで複数のネコの画像を管理
    None, ・・・何もない値を意味する
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png")
]

cvs.create_image(456,384,image=bg)  ・・・キャンバス上に背景を描く
game_main()・・・メインの処理を行う関数を呼び出す
root.mainloop()・・・ウィンドウを表示する
```

ブロックを落下させるアルゴリズム

「Run Module」またはF5キーを押してプログラムを実行する。

ブロックが落下することを確認しよう



クリックしてブロックを置く

クリックした位置にネコを置く処理を組み込む。

```
import tkinter  ・・・・tkinterモジュールの呼出し
import random   ・・・・randomモジュールの呼出し

cursor_x = 0    ・・・・カーソルの横方向の位置(左から何マス目にあるか)
cursor_y = 0    ・・・・カーソルの縦方向の位置(上から何マス目にあるか)
mouse_x = 0     ・・・・マウスポインタのX座標
mouse_y = 0     ・・・・マウスポインタのY座標
mouse_c = 0     ・・・・マウスボタンをクリックした時の変数（フラグ）

def mouse_move(e):  ・・・・マウスを動かした時に実行する関数
    global mouse_x , mouse_y ・・・・これらをグローバル変数として扱うと宣言
    mouse_x = e.x   ・・・・mouse_xにマウスポインタのX座標を代入
    mouse_y = e.y   ・・・・mouse_yにマウスポインタのY座標を代入

def mouse_press(e):・・・・マウスボタンをクリックした時に実行する関数
    global mouse_c  ・・・・この変数をグローバル変数として扱うと宣言
    mouse_c = 1     ・・・・mouse_cに1を代入
```

次のページに続く

クリックしてブロックを置く

```
neko=[                    ・・・・マス目を管理する二次元リスト
[1,0,0,0,0,0,7,7],
[0,2,0,0,0,0,7,7],
[0,0,3,0,0,0,0,0],
[0,0,0,4,0,0,0,0],
[0,0,0,0,5,0,0,0],
[0,0,0,0,0,6,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,1,2,3,4,5,6]
]
def draw_neko():  ・・・・ネコを表示する関数
    for y in range(10):     ・・・繰り返し yは0から9まで１ずつ増える
        for x in range(8): ・・・繰り返し xは0から7まで１ずつ増える
            cvs.create_image(x*72+60,y*72+60,image=img_neko[neko[y][x]],tag="NEKO")
def drop_neko():・・ネコを落下させる関数     ・・・img_nekoに指定した画像を表示する
    for y in range(8,-1,-1): ・・・・繰り返し yは8から0まで1ずつ減る
        for x in range(8):   ・・・・繰り返し xは0から7まで1ずつ増える
            if neko[y][x] != 0 and neko[y+1][x]==0: ・・ネコのあるマスの下
                neko[y+1][x] = neko[y][x] が空白なら空白にネコを入れ、元のマ
                neko[y][x] = 0                    スは空白にする
```

次のページに続く

クリックしてブロックを置く

```
def game_main():・・・・メインの処理を行う関数
    global cursor_x,cursor_y,mouse_c ・・これらをグローバル変数として扱うと宣言
    drop_neko() ・・・・ネコを落下させる関数を呼び出す
    if 24 <= mouse_x and mouse_x < 24+72*8 and 24 <= mouse_y and mouse_y < 24+72*10:
        cursor_x = int((mouse_x-24)/72)・・マウスポインタの座標が盤面上であればポインタのX座標からカーソルの横の位置を計算、ポインタのY座標からカーソルの縦の位置を計算
        cursor_y = int((mouse_y-24)/72)
        if mouse_c ==1:・・マウスボタンをクリックしたらクリックしたフラグを解除
            mouse_c = 0
            neko[cursor_y][cursor_x] = random.randint(1,6)・・カーソルのマスにランダムにネコを配置
    cvs.delete("CURSOR") ・・カーソルを消す
    cvs.create_image(cursor_x*72+60,cursor_y*72+60,image = cursor,tag ="CURSOR")
    cvs.delete("NEKO")・ネコの画像を削除 ・・新たな位置にカーソルを表示する
    draw_neko()・・ネコを表示
    root.after(100,game_main)・・0.1秒後に再びメインの処理を実行

root=tkinter.Tk()・・・・ウィンドウのオブジェクトを作る
root.title("クリックしてネコを置く")・・・・タイトルを指定
root.resizable(False,False) ・・・・ウィンドウサイズを変更できないようにする
root.bind("<Motion>",mouse_move)・・・・マウスが動いた時に実行する関数を指定
root.bind("<ButtonPress>",mouse_press)・マウスボタンをクリックした時に実行する関数を指定
cvs=tkinter.Canvas(root,width=912,height=768)・・キャンバスの部品を作る
cvs.pack()・・キャンバスを配置する
```

次のページに続く

クリックしてブロックを置く

```
bg = tkinter.PhotoImage(file="neko_bg.png")・・・背景画像の読み込み
cursor = tkinter.PhotoImage(file="neko_cursor.png")・・カーソル画像の読み込み
img_neko=[                                          ・・・リストで複数のネコの画像を管理
    None, ・・・何もない値を意味する
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png")
]

cvs.create_image(456,384,image=bg) ・・・キャンバス上に背景を描く
game_main() ・・・メインの処理を行う関数を呼び出す
root.mainloop() ・・・ウィンドウを表示する
```

ブロックがそろったことを判定する

ブロックがそろったことを判定するアルゴリズム。同じブロックが横に３つ並んだ状態を判定するプログラムを作成する。

３つ並んだことを調べる

if文を使うことで、もし３つのブロックがそろったならという条件を指定することができる

```
if neko[y][x-1] == neko[y][x] and neko[y][x+1]==neko[y][x]:
```

今回はネコが３つそろった場合、ネコを肉球に変える。そのため、if文の条件が真のとき、以下を記述する。

```
neko[y][x-1]=7 neko[y][x]=7 neko[y][x+1]=7
```

append()命令について理解する

append()命令を使うと既存のリストに単一の要素を追加することができる。append()命令は二次元リストでも使用可能。

appendとは「添える」「付加する」という意味があり、リストに要素を追加するためのメソッド

使用例① 単一の要素を追加する

```
a = [1,2,3]
a.append(4)
print(a)
```

```
[1, 2, 3, 4]
>>>
```

使用例② リストを追加する

```
a=[1,2,3]
a.append([4,5,6])
print(a)
```

```
[1, 2, 3, [4, 5, 6]]
>>>
```

複数の要素を追加する場合、リストの形で加える

append()命令について理解する

使用例③ for文を使用して複数の要素を追加する

```
a=[1,2,3]
for i in range(3):
    a.append(4)
print(a)
```

```
[1, 2, 3, 4, 4, 4]
>>>
```

使用例④ for文を使用して複数の要素を追加する

```
a=[1,2,3]
b=3
for i in range(3):
    b=b+1
    a.append(b)
print(a)
```

```
[1, 2, 3, 4, 5, 6]
>>>
```

append()命令について理解する

使用例⑤ for文を使用して空のリストにリストを追加する

```
a=[]
for i in range(5):
    a.append([0,0,0])
print(a)
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>>
```

ブロックがそろったことを判定する

クリックした位置にネコを置く処理を組み込む。

```
import tkinter  ・・・・tkinterモジュールの呼出し
import random   ・・・・randomモジュールの呼出し

cursor_x = 0    ・・・・カーソルの横方向の位置(左から何マス目にあるか)
cursor_y = 0    ・・・・カーソルの縦方向の位置(上から何マス目にあるか)
mouse_x = 0     ・・・・マウスポインタのX座標
mouse_y = 0     ・・・・マウスポインタのY座標
mouse_c = 0     ・・・・マウスボタンをクリックした時の変数（フラグ）

def mouse_move(e):  ・・・・マウスを動かした時に実行する関数
    global mouse_x , mouse_y  ・・・・これらをグローバル変数として扱うと宣言
    mouse_x = e.x   ・・・・mouse_xにマウスポインタのX座標を代入
    mouse_y = e.y   ・・・・mouse_yにマウスポインタのY座標を代入

def mouse_press(e):・・・・マウスボタンをクリックした時に実行する関数
    global mouse_c  ・・・・この変数をグローバル変数として扱うと宣言
    mouse_c = 1     ・・・・mouse_cに1を代入

neko=[]   ・・・・マス目を管理する二次元リスト
for i in range(10):    ・・・・繰り返しとappend()命令でリストを初期化する
    neko.append([0,0,0,0,0,0,0,0])
```

次のページに続く

ブロックがそろったことを判定する

```
def draw_neko():  ・・・・ネコを表示する関数
    for y in range(10):    ・・・繰り返し yは0から9まで１ずつ増える
        for x in range(8): ・・・繰り返し xは0から7まで１ずつ増える
            cvs.create_image(x*72+60,y*72+60,image=img_neko[neko[y][x]],tag="NEKO")
                                        ・・・二次元リストから条件にあった画像を表示する
def yoko_neko():   ・・・ネコが横に3つ並んだか調べる関数
    for y in range(10): ・・・繰り返し yは0から9まで１ずつ増える
        for x in range(1,7):・・・繰り返し xは1から6まで１ずつ増える
            if neko[y][x]>0:  ・・・マスにネコが配置されていて
                if neko[y][x-1] == neko[y][x] and neko[y][x+1]==neko[y][x]:
                    neko[y][x-1]=7   左右が同じネコなら
                    neko[y][x]=7     それらのマスを肉球に変える
                    neko[y][x+1]=7
```

次のページに続く

ブロックがそろったことを判定する

```
def game_main():・・・・メインの処理を行う関数
    global cursor_x,cursor_y,mouse_c ・・これらをグローバル変数として扱うと宣言
    if 660 <= mouse_x and mouse_x < 840 and 100 <= mouse_y < 160 and mouse_c == 1:
        mouse_c = 0・・風船内の「変換」という文字をクリックしたらクリックしたフラグを削除
        yoko_neko()  ・・・横に並んだか調べる関数を呼び出す
    if 24 <= mouse_x and mouse_x < 24+72*8 and 24 <= mouse_y and mouse_y < 24+72*10:
        cursor_x = int((mouse_x-24)/72)・・マウスポインタの座標が盤面上であればポ
        cursor_y = int((mouse_y-24)/72)インタのX座標からカーソルの横の位置を計算、
        if mouse_c ==1:                ポインタのY座標からカーソルの縦の位置を計算
            mouse_c = 0 ・・マウスボタンをクリックしたらクリックしたフラグを解除
            neko[cursor_y][cursor_x] = random.randint(1,2)・カーソルのマスにラ
    cvs.delete("CURSOR")・・カーソルを消す                ンダムにネコを配置
    cvs.create_image(cursor_x*72+60,cursor_y*72+60,image = cursor,tag ="CURSOR")
    cvs.delete("NEKO")・ネコの画像を削除          ・・新たな位置にカーソルを表示する
    draw_neko()・・ネコを表示
    root.after(100,game_main)・・0.1秒後に再びメインの処理を実行

root=tkinter.Tk()  ・・・・ウィンドウのオブジェクトを作る
root.title("横に３つ並んだか")  ・・・・タイトルを指定
root.resizable(False,False)  ・・・・ウィンドウサイズを変更できないようにする
root.bind("<Motion>",mouse_move)  ・・・・マウスが動いた時に実行する関数を指定
root.bind("<ButtonPress>",mouse_press)・マウスボタンをクリックした時に実行する関数を指定
```

次のページに続く

ブロックがそろったことを判定する

```
cvs=tkinter.Canvas(root,width=912,height=768)・・キャンバスの部品を作る
cvs.pack()・・キャンバスを配置する

bg = tkinter.PhotoImage(file="neko_bg.png")・・・背景画像の読み込み
cursor = tkinter.PhotoImage(file="neko_cursor.png")・・カーソル画像の読み込み
img_neko=[                                          ・・・リストで複数のネコの画像を管理
    None,・・・何もない値を意味する
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png")
]

cvs.create_image(456,384,image=bg)・・・キャンバス上に背景を描く
cvs.create_rectangle(660,100,840,160,fill="white")・・・風船内に枠を描く
cvs.create_text(750,130,text="変換",fill="red",font=("Times New Roman",30))
                                                    ・・・変換と表示する
game_main()・・・メインの処理を行う関数を呼び出す
root.mainloop()・・・ウィンドウを表示する
```

ブロックがそろったことを判定する

「Run Module」またはF5キーを押してプログラムを実行する。

３つそろった状態で「変換」ボタンを押すとブロックが肉球になることを確認しよう

この判定の問題点について考える

この方法で横に３つ並んだことを判定できることが分かった。同様に縦に３つ並んだかを判定し、斜めにも３つ並んだかを判定すれば良いように思える。しかし、この方法には問題点があることを知っておこう。

問題点①

４つ、５つ、７つ、８つ並んだ状態では判定できないマスがでる。

右端が１つ変換されず、残ってしまう。

この判定の問題点について考える

問題点②
横に並んだかを判定した後、縦に並んだかを判定すると、正しく判定できないことがある
まず、縦の並びも判定できるプログラムをつくる

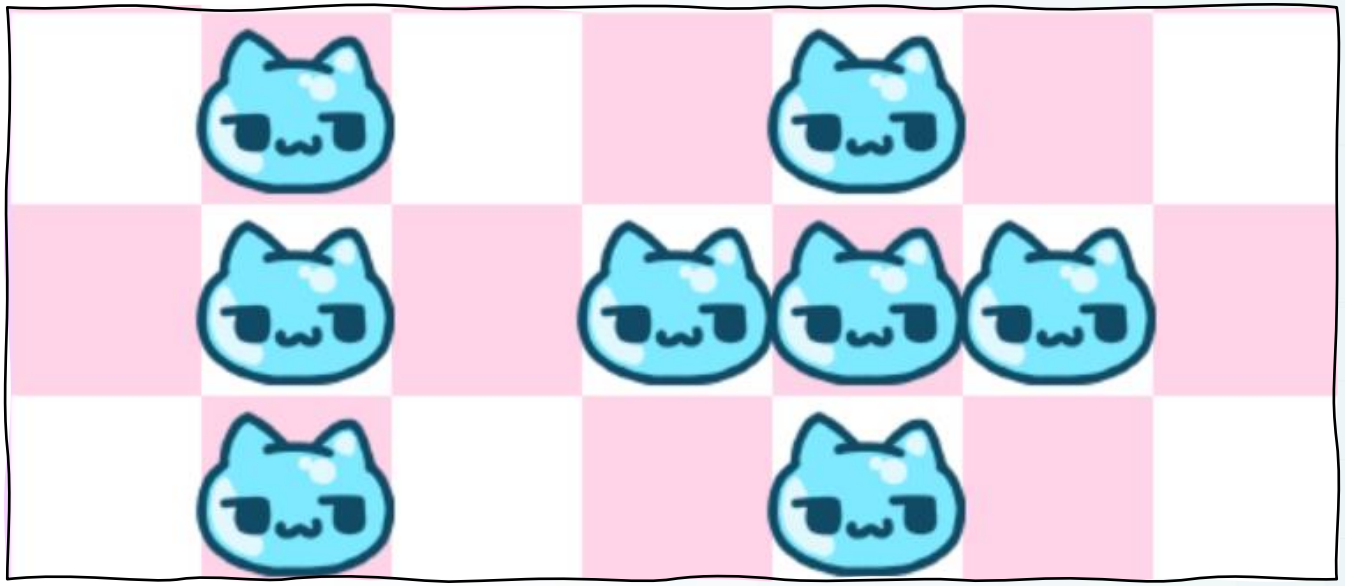
```
def yoko_neko():
    for y in range(10):
        for x in range(1,7):
            if neko[y][x]>0:
                if neko[y][x-1] == neko[y][x] and neko[y][x+1]==neko[y][x]:
                    neko[y][x-1]=7
                    neko[y][x]=7
                    neko[y][x+1]=7
```

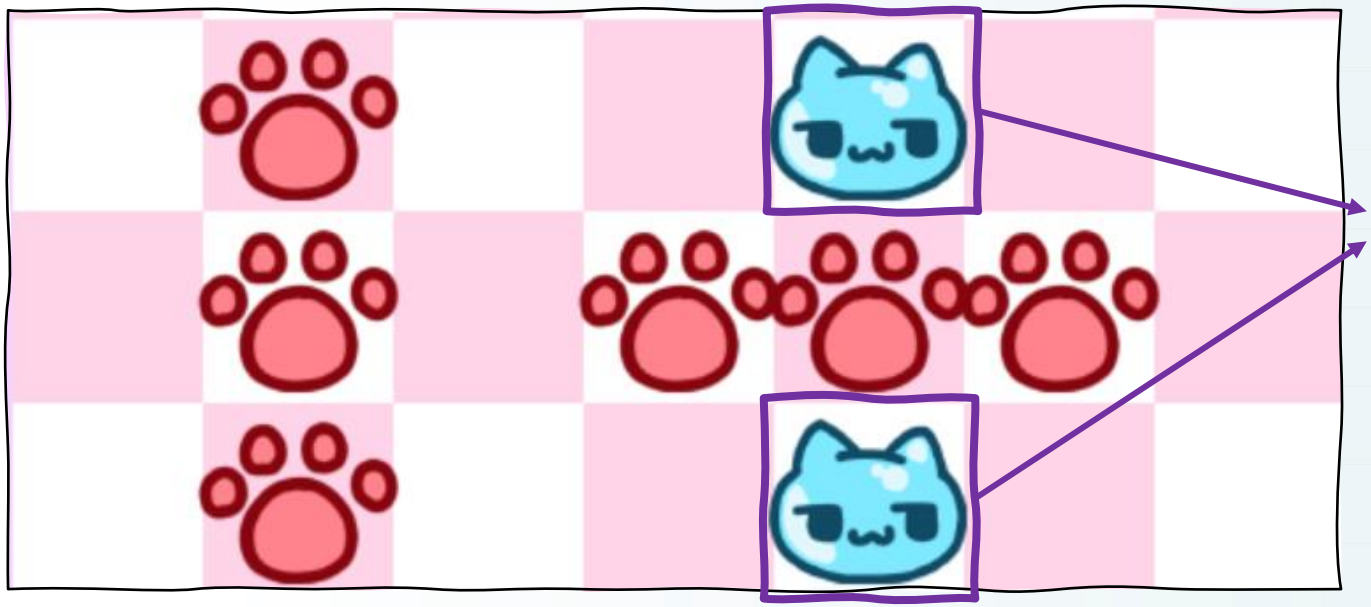
横に３つ並んだか

```
    for y in range(1,9):
        for x in range(8):
            if neko[y][x]>0:
                if neko[y-1][x] == neko[y][x] and neko[y+1][x]==neko[y][x]:
                    neko[y-1][x]=7
                    neko[y][x]=7
                    neko[y+1][x]=7
```

縦に３つ並んだか

この判定の問題点について考える





十字に並んでいると先に横方向を判定するため、縦方向は判定されない

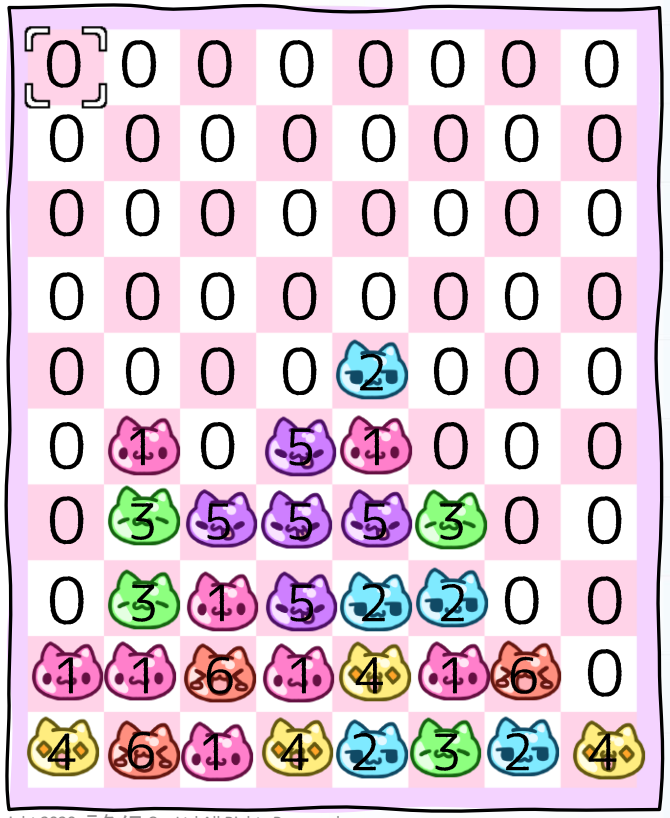
正しいアルゴリズムを考える

問題を解決し、縦、横、斜めに３つ以上ブロックが揃ったことを正確に判定するアルゴリズムを考える。

①判定用のリストを用紙、そこにマス目のデータをコピーする

②判定用のリストを調べて同じネコが３つ並んでいるかを確認し、並んでいる場合はゲーム用のリストの値を変更する

ゲーム用のリストneko[][]の値



データをコピーしたリスト

```
check=[
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,0,0,0,0],
[0,0,0,0,2,0,0,0],
[0,1,0,5,1,0,0,0],
[0,3,5,5,5,3,0,0],
[0,3,1,5,2,2,0,0],
[1,1,6,1,4,1,6,0],
[4,6,1,4,2,3,2,4]
]
```

縦、横を同時に判定させるためにデータをコピーしたリストを作成する

正しいアルゴリズムを考える

縦、横、斜めに3つ以上、ネコが揃ったことを正確に判定する

```
import tkinter ・・・・tkinterモジュールの呼出し
import random ・・・・randomモジュールの呼出し

cursor_x = 0 ・・・・カーソルの横方向の位置(左から何マス目にあるか)
cursor_y = 0 ・・・・カーソルの縦方向の位置(上から何マス目にあるか)
mouse_x = 0 ・・・・マウスポインタのX座標
mouse_y = 0 ・・・・マウスポインタのY座標
mouse_c = 0 ・・・・マウスボタンをクリックした時の変数（フラグ）

def mouse_move(e): ・・・・マウスを動かした時に実行する関数
    global mouse_x , mouse_y ・・・・これらをグローバル変数として扱うと宣言
    mouse_x = e.x ・・・・mouse_xにマウスポインタのX座標を代入
    mouse_y = e.y ・・・・mouse_yにマウスポインタのY座標を代入

def mouse_press(e):・・・・マウスボタンをクリックした時に実行する関数
    global mouse_c ・・・・この変数をグローバル変数として扱うと宣言
    mouse_c = 1 ・・・・mouse_cに1を代入
```

次のページに続く

正しいアルゴリズムを考える

```
neko=[]・・・・マス目を管理する二次元リスト
check=[]・・・・判定用の二次元リスト
for i in range(10):          ・・・・繰り返しとappend()命令でリストを初期化する
    neko.append([0,0,0,0,0,0,0,0])
    check.append([0,0,0,0,0,0,0,0])

def draw_neko():・・・・ネコを表示する関数
    for y in range(10):・・・繰り返し yは0から9まで1ずつ増える
        for x in range(8):・・・繰り返し xは0から7まで1ずつ増える
            cvs.create_image(x*72+60,y*72+60,image=img_neko[neko[y][x]],tag="NEKO")
                                        ・・・二次元リストから条件にあった画像を表示する
def check_neko():・・・ネコが縦、横、斜めに3つ以上並んだか調べる関数
    for y in range(10):・・・繰り返し yは0から9まで1ずつ増える
        for x in range(8):  ・・・繰り返し xは0から7まで1ずつ増える
            check[y][x]=neko[y][x] ・・・判定用のリストにネコの値を入れる

    for y in range(1,9):・・・繰り返し yは1から8まで1ずつ増える
        for x in range(8):   ・・・繰り返し xは0から7まで1ずつ増える
            if check[y][x]>0:・空白マスどうしは同じ値が入っていると認識させない
                if check[y-1][x] == check[y][x] and check[y+1][x] == check[y][x]:
                    neko[y-1][x]=7 ・・・マスにネコが配置されていて上下が
                    neko[y][x]=7    同じネコならそれらのマスを肉球に変える
                    neko[y+1][x]=7
```

次のページに続く

正しいアルゴリズムを考える

```
for y in range(10):  ・・・繰り返し yは0から9まで1ずつ増える
    for x in range(1,7):  ・・繰り返し xは1から6まで1ずつ増える
        if check[y][x]>0:  ・空白マスどうしは同じ値が入っていると認識させない
            if check[y][x-1] == check[y][x] and check[y][x+1] == check[y][x]:
                neko[y][x-1]=7  ・・・マスにネコが配置されていて左右が
                neko[y][x]=7      同じネコならそれらのマスを肉球に変える
                neko[y][x+1]=7

for y in range(1,9):  ・・・繰り返し yは1から8まで1ずつ増える
    for x in range(1,7):  ・・・繰り返し xは1から6まで1ずつ増える
        if check[y][x]>0:  ・空白マスどうしは同じ値が入っていると認識させない
            if check[y-1][x-1] == check[y][x] and check[y+1][x+1] == check[y][x]:
                neko[y-1][x-1]=7  ・・・マスにネコが配置されていて左上
                neko[y][x]=7        と右下が同じネコならそれらのマスを肉
                neko[y+1][x+1]=7    球に変える
            if check[y+1][x-1] == check[y][x] and check[y-1][x+1] == check[y][x]:
                neko[y+1][x-1]=7  ・・・マスにネコが配置されていて左下
                neko[y][x]=7        と右上が同じネコならそれらのマスを肉
                neko[y-1][x+1]=7    球に変える
```

次のページに続く

正しいアルゴリズムを考える

```
def game_main():・・・・メインの処理を行う関数
    global cursor_x,cursor_y,mouse_c・・これらをグローバル変数として扱うと宣言
    if 660 <= mouse_x and mouse_x < 840 and 100 <= mouse_y < 160 and mouse_c == 1:
        mouse_c = 0 ・風船内の「変換」という文字をクリックしたらクリックしたフラグを削除
        check_neko()・・・ネコが並んだか調べる関数を呼び出す
    if 24 <= mouse_x and mouse_x < 24+72*8 and 24 <= mouse_y and mouse_y < 24+72*10:
        cursor_x = int((mouse_x-24)/72) ・・マウスポインタの座標が盤面上であればポインタのX座標からカーソルの横の位置を計算、ポインタのY座標からカーソルの縦の位置を計算
        cursor_y = int((mouse_y-24)/72)
        if mouse_c ==1:・・マウスボタンをクリックしたらクリックしたフラグを解除
            mouse_c = 0
            neko[cursor_y][cursor_x] = random.randint(1,2)
    cvs.delete("CURSOR")・カーソルを消す ・カーソルのマスにランダムにネコを配置
    cvs.create_image(cursor_x*72+60,cursor_y*72+60,image = cursor,tag ="CURSOR")
    cvs.delete("NEKO")・・ネコの画像を削除 ・・新たな位置にカーソルを表示する
    draw_neko()・・ネコを表示
    root.after(100,game_main)・・0.1秒後に再びメインの処理を実行

root=tkinter.Tk()・・・・ウィンドウのオブジェクトを作る
root.title("縦、横、斜めに３つ以上並んだか")・・・・タイトルを指定
root.resizable(False,False)・ウィンドウサイズを変更できないようにする
```

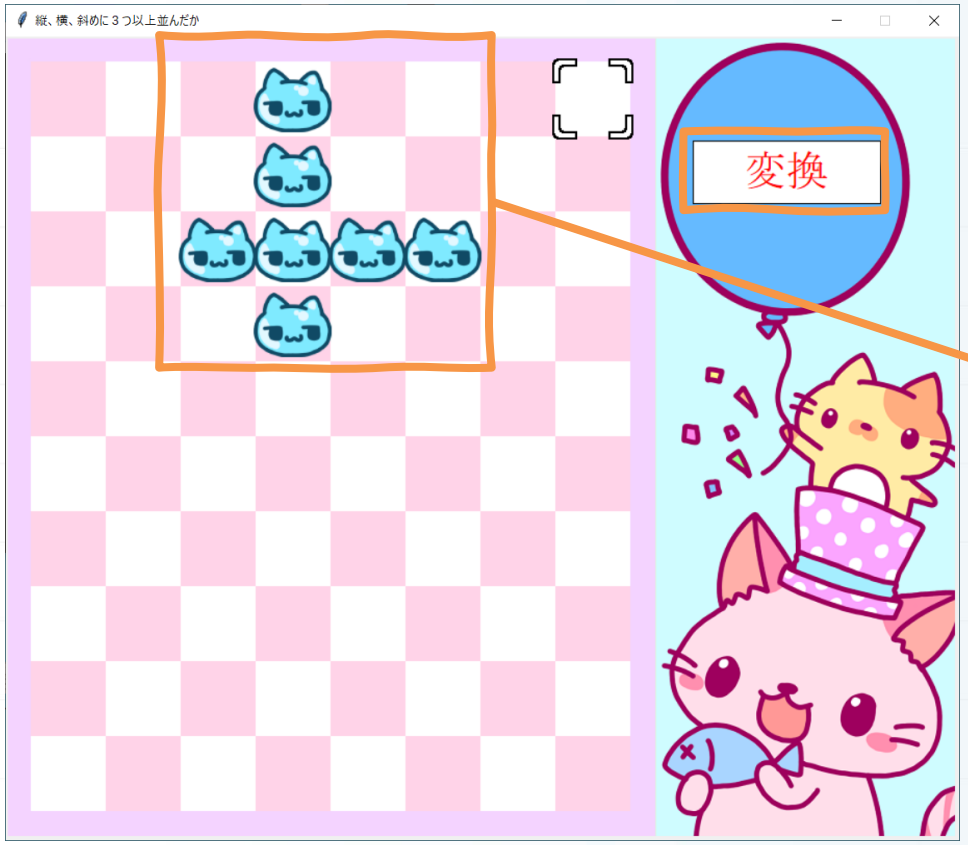
次のページに続く

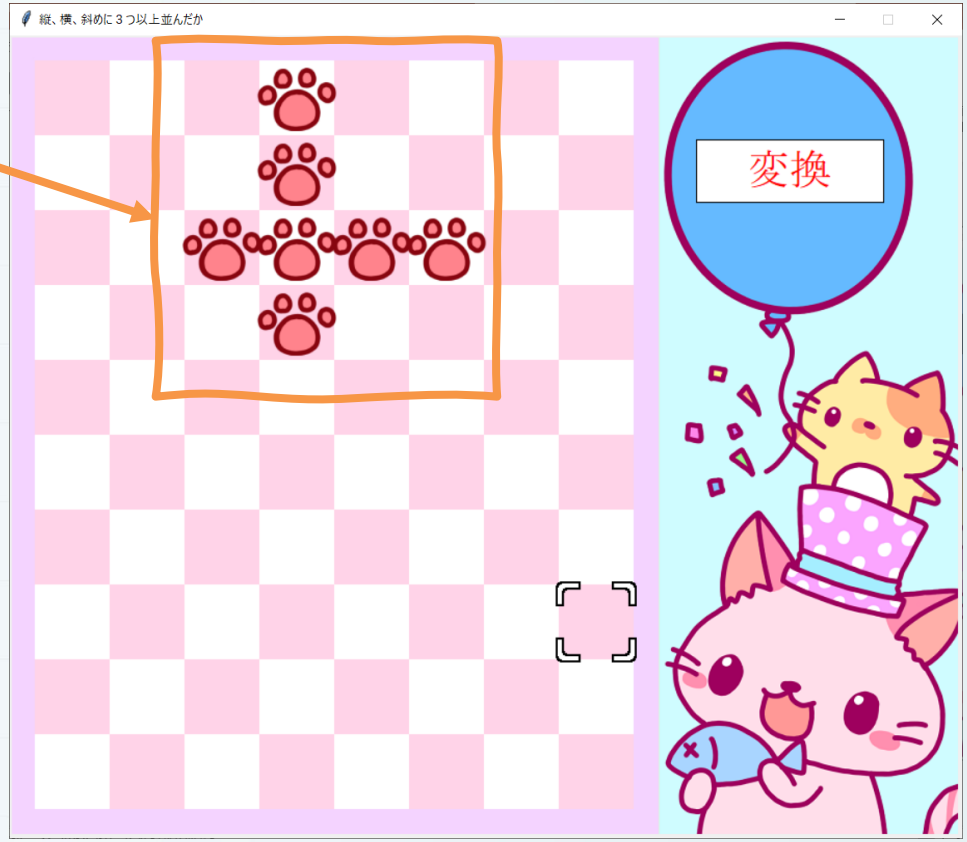
正しいアルゴリズムを考える

```
root.bind("<Motion>",mouse_move)・・マウスが動いた時に実行する関数を指定
root.bind("<ButtonPress>",mouse_press)・マウスボタンをクリックした時に実行する関数を指定
cvs=tkinter.Canvas(root,width=912,height=768)・・キャンバスの部品を作る
cvs.pack()・・キャンバスを配置する
bg = tkinter.PhotoImage(file="neko_bg.png")・・・背景画像の読み込み
cursor = tkinter.PhotoImage(file="neko_cursor.png")
img_neko=[・・・リストで複数のネコの画像を管理          ・・カーソル画像の読み込み
    None,・・・何もない値を意味する
    tkinter.PhotoImage(file="neko1.png"),
    tkinter.PhotoImage(file="neko2.png"),
    tkinter.PhotoImage(file="neko3.png"),
    tkinter.PhotoImage(file="neko4.png"),
    tkinter.PhotoImage(file="neko5.png"),
    tkinter.PhotoImage(file="neko6.png"),
    tkinter.PhotoImage(file="neko_niku.png")
]
cvs.create_image(456,384,image=bg)・・・キャンバス上に背景を描く
cvs.create_rectangle(660,100,840,160,fill="white")・・・風船内に枠を描く
cvs.create_text(750,130,text="変換",fill="red",font=("Times New Roman",30))
game_main()・・・メインの処理を行う関数を呼び出す          ・・・変換と表示する
root.mainloop()・・・ウィンドウを表示する
```

ブロックがそろったことを判定する

「Run Module」またはF5キーを押してプログラムを実行する。





３つ以上そろった状態で「変換」ボタンを押すとブロックがすべて肉球になることを確認しよう

復習＆チャレンジ

ここまで習ったことをScratchでもできるかチャレンジしてみよう。
その過程でScratchでできること、Pythonでないとできないことを整理してみよう。

メモ

プログラミング教室のテクノロ

なまえ：