
# 5日でできる!
# Python ゲームプログラミング
## 入門

人気付録の続編!

中島 省吾（メディアプラネット）著

Pythonで一番簡単にゲームを作れるPygame Zeroを使って神経衰弱やスロットゲーム、荷物運びパズルゲームを作ろう

日経ソフトウエア
NIKKEI SOFTWARE
2021年5月号 第2付録

特別付録② 5日でできる! Pythonゲームプログラミング入門

日経ソフトウエア
NIKKEI SOFTWARE

1日目

# PythonとPygame Zeroを復習しよう

# 1日目 Part 1 Pythonの導入

地元の公立中学校に通う幼なじみの錦田達也(タツヤ)と佐藤麗(レイ)は、同じプログラミング教室の「Pythonゲームコース」に通うほどの仲良し。ところが、新型コロナウイルスの蔓延で、プログラミング教室は休校になってしまった。しばらく、プログラミングから離れていた2人だが、ついに教室がオンライン授業を開始。自宅のパソコンで「続・Pythonゲームコース」の授業を受講することになった。

なお、2人がPythonゲームコースを受講している様子は、日経ソフトウエア2020年5月号特別付録小冊子「5日でできる! Pythonでゲーム作成入門」に収録しています。今回のお話は、「5日でできる! Pythonでゲーム作成入門」の続編です。

**先生** こんにちは〜。聞こえるかな〜。2人とも元気にしてたかぁ。

**タツヤ** こんにちは〜。なんだか、オンラインは緊張すんなぁ。

**レイ** おひさ〜、みんな。元気だよ〜。

**先生** そうかぁ。それは良かった。教室がお休みの間、プログラミングの実力は上がったかな?

**タツヤ** そりゃ…、もう…、ぜんぜん、やってませーん。

**レイ** 違うのよ。センセイ、聞いて!

**先生** ん、どうした?

**レイ** 新しいゲーミングパソコンを買ったの。グラフィックスがめっちゃ綺麗で、デュアルディスプレイなの。

**タツヤ** そしたらさぁ、レイがオンラインゲームに誘ってきて…。

**レイ** なによ〜、先にゲーミングパソコン買ったのはタツヤじゃない。

**先生** ハイハイ。言い訳はそれくらいにして。つまり、パソコンが新しくなって、Pythonのプログラミング環境もなくなってしまった、ってことだな。

**タツヤ レイ** は〜い。

**先生** それでは、Pythonをパソコンにインストールするところから始めよう。Windows 10では、「コマンドプロンプト」や「Windows PowerShell」で「python」と入力すると、「Microsoft Store」が起動するようになっている。ちなみに、これを煩わしく思う場合は、「設定」→「アプリ」の画面で「アプリ実行エイリアス」をクリックして、「アプリ インストーラー（python.exe）」を「オフ」にしよう（**図1**）。これで、Microsoft Storeは起動しなくなる。

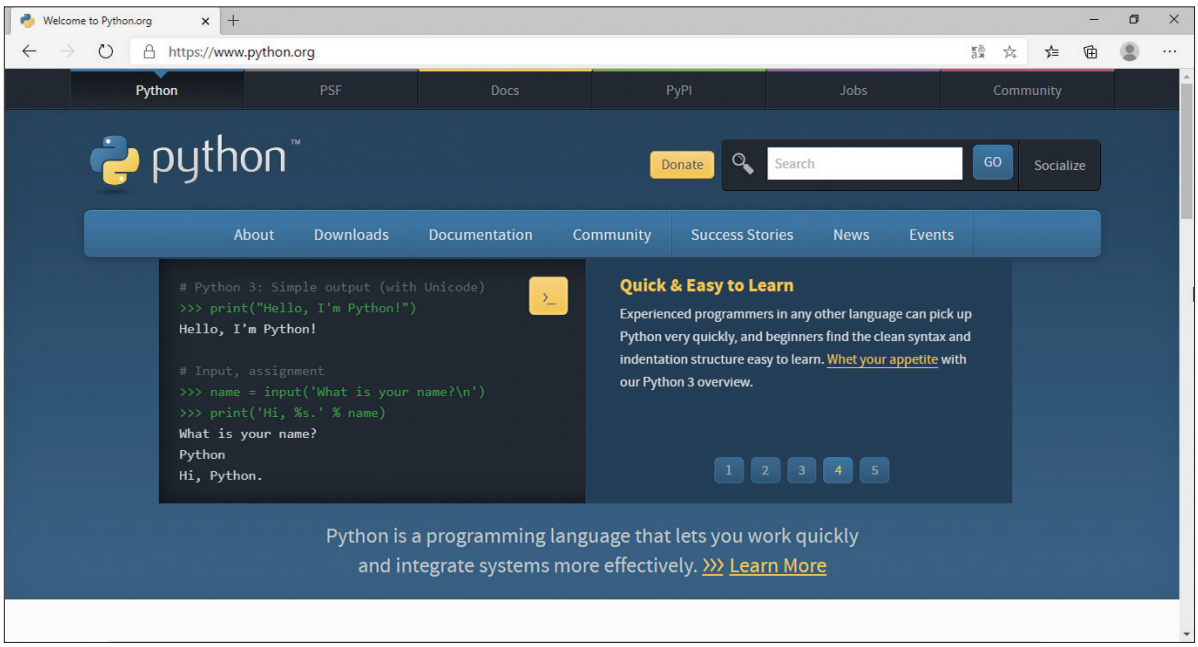
**図1●アプリ実行エイリアス**


**レイ** 「オフ」に、と。

**先生** 今回は、Microsoft Storeのインストーラーは使わずに、“Pythonの公式サイト”か

らインストーラーをダウンロードして、Pythonのプログラミング環境を導入する（**図2**）。前回のPythonゲームコースのときにインストールしたPythonのバージョンは3.8.1だったけど、今回は3.9.1になっている（2021年2月時点）。

**図2●Pythonの公式サイト（https://www.python.org/）**



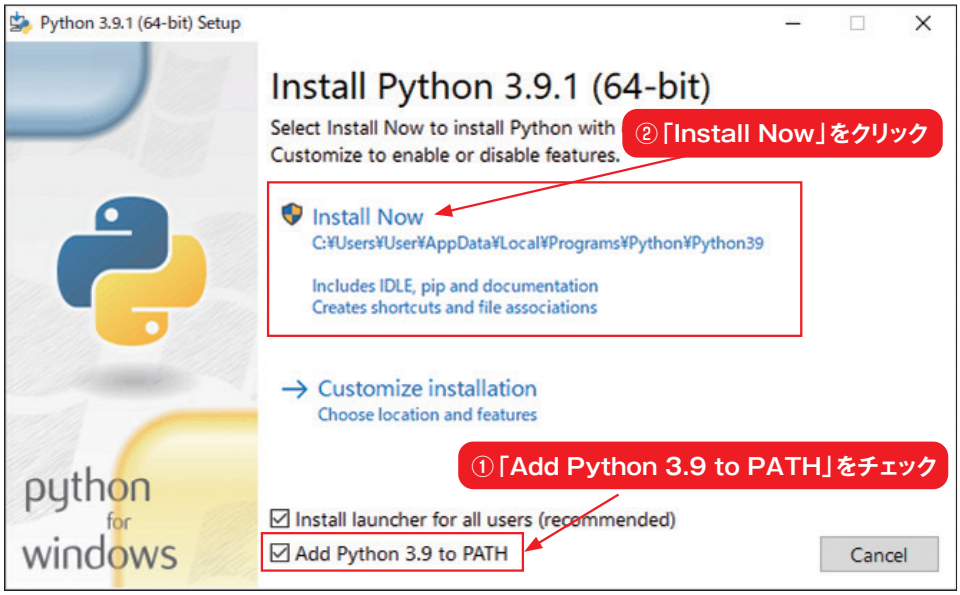

**レイ** ちょっとだけバージョンが上がっているのね。

**先生** そうだね。でも、Python自体に大きな違いはないよ。ただ、後で説明するけど、ちょっと面倒になったことがある。

**レイ** 気になる…。

**先生** Windows用のインストーラーをダウンロードしたら、早速実行しよう。このとき、インストーラーの最初の画面に表示される「Add Python 3.9 to PATH」にチェックを入れておくと、どのフォルダーからでもPythonのツールを起動できて便利だ（**図3**）。

**図3●Windows用のインストーラー**




**レイ** インストール完了!

**タツヤ** スタートメニューにもPythonのアイコンが出たよ。

**先生** それでは、Pythonが動くかどうか確認しよう。今回もPythonのプログラムは、標準でインストールされる「IDLE」で作るぞ。Windowsの「スタートボタン」をクリックしたら、「IDLE(Python 3.9 64-bit)」を選択しよう(**図4**)。

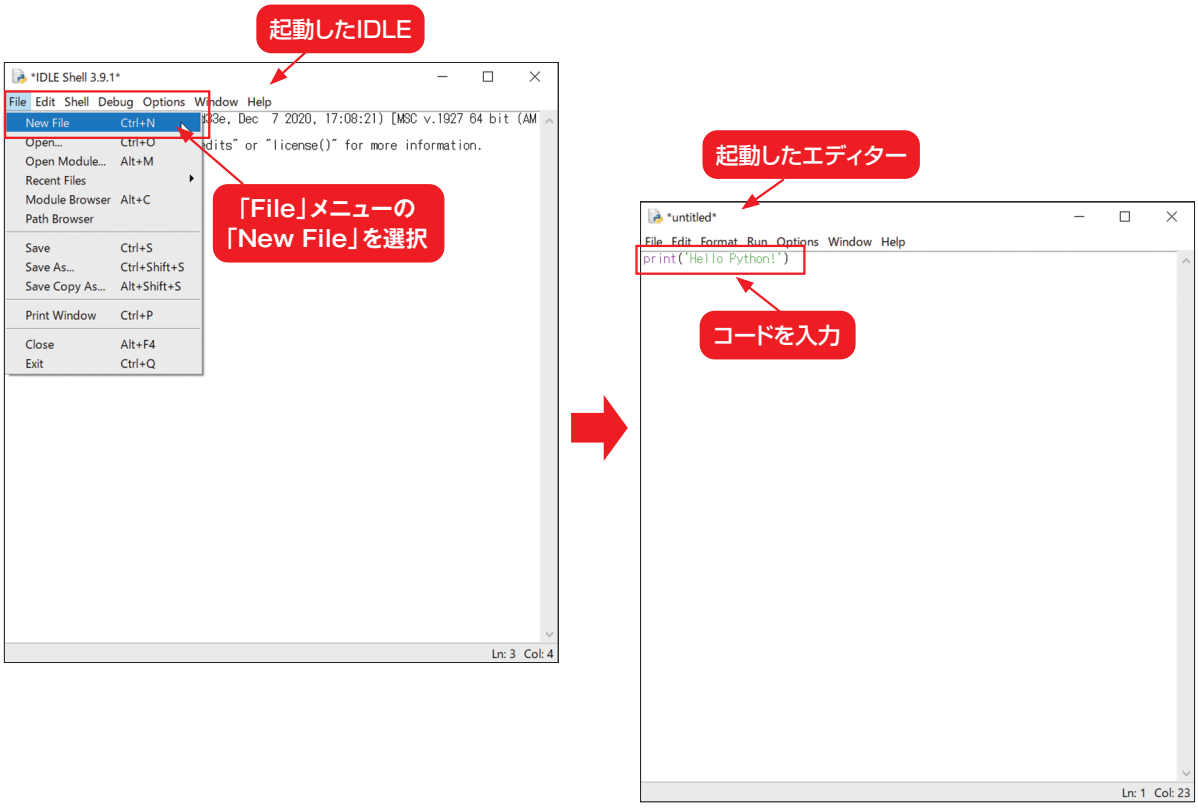
**図4●「IDLE(Python 3.9 64-bit)」を起動**

**レイ** 起動!

**先生** IDLEが起動したら、「File」メニューから「New　File」を選択して、エディターを起動。次のコードを入力する(**図5**)。

```
print('Hello Python!')
```

**図5●コードを入力**




**タツヤ** printは、文字を表示する関数だったな。

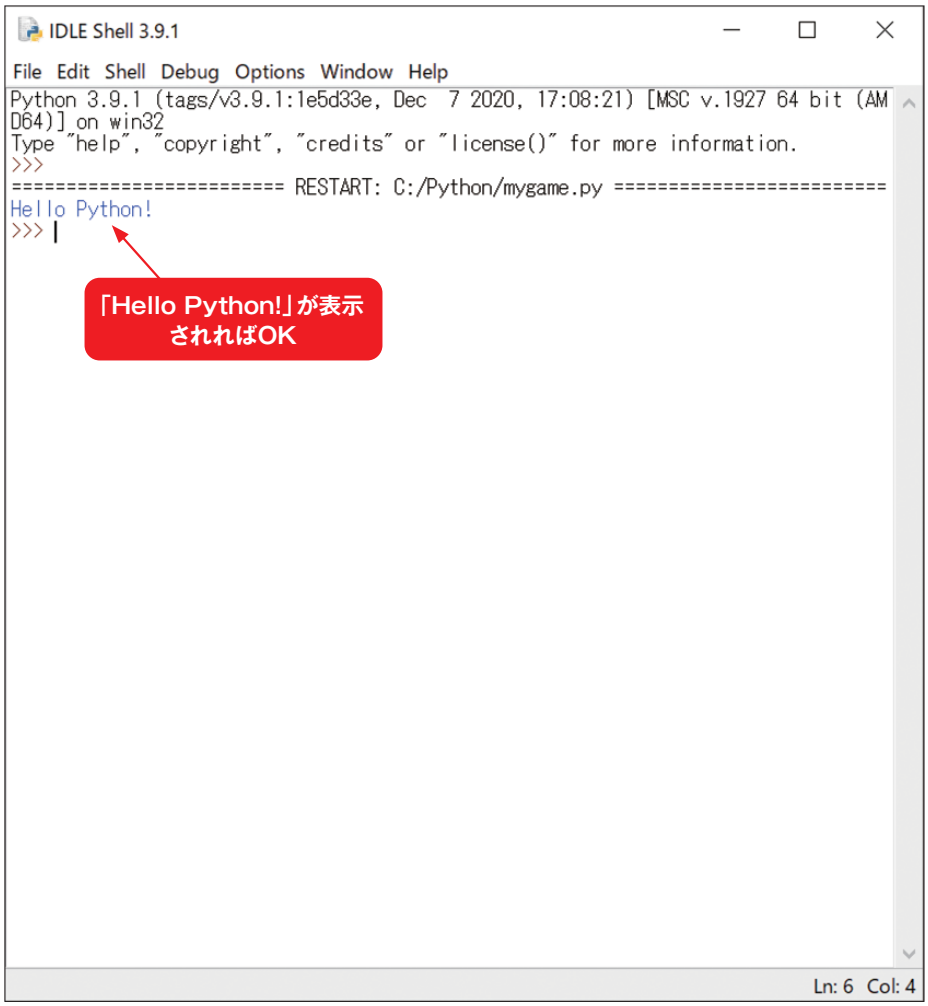
**先生** そう。コードを入力できたら、ファイルに保存しよう。メニューの「File」→「Save As...」を選択して保存用のダイアログを表示する(**図6**)。ここで、作業用のフォルダーを作っておこう。Cドライブの直下に「Python」フォルダーを作成する。そしてこの中にファイルを保存する。ファイル名は「mygame.py」にしよう。

**図6●Pythonフォルダーの作成とソースファイルの保存**


**タツヤ** 保存しましたよ。

**先生** IDLEのメニューの「Run」→「Run　Module」をクリックするか、IDLEのエディターを前面にして「F5」キーを押そう。プログラムが実行されて「Hello Python!」と表示されたら成功だ（**図7**）。Pythonは正常に動作している。

**図7●プログラムを実行**




**レイ** 表示できた。

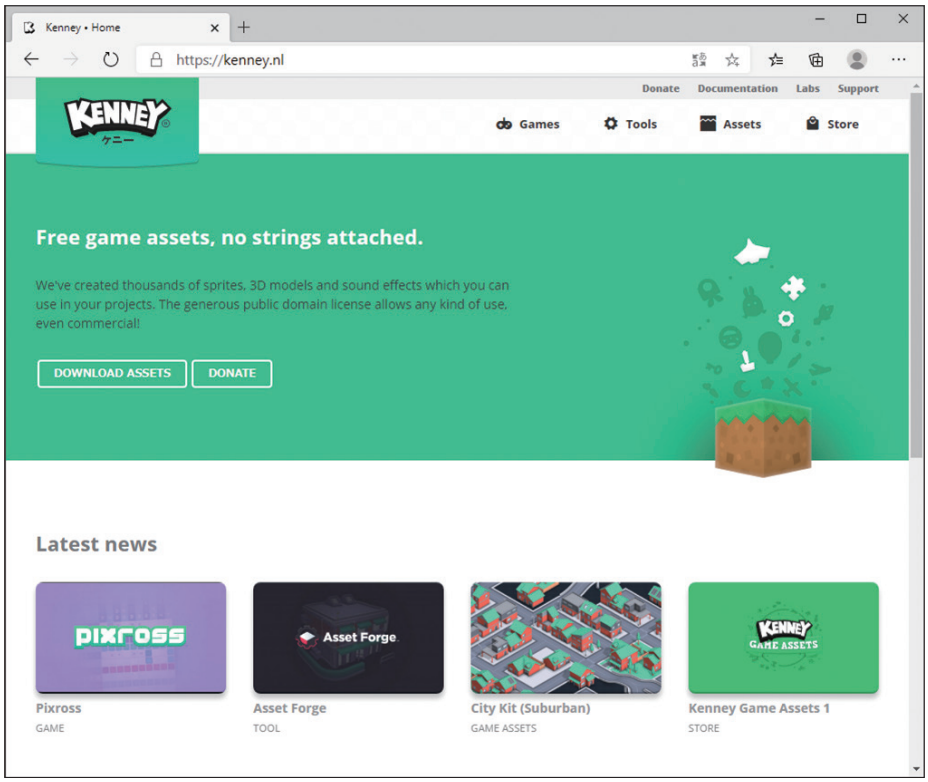
**タツヤ** 簡単、簡単。

**先生** 次は、プログラムで使用する画像をダウンロードしよう。

# 1日目 Part 2 ケニーのサイト

**先生** Pythonをインストールできたら、ゲームで使う画像をダウンロードしよう。今回も「ケニー」（https://kenney.nl/）のサイトにある画像を使おう（**図1**）。

**図1●ケニーのサイト（https://kenney.nl/）。無償で利用できるゲーム向けの画像がたくさんある**




**レイ** このサイト、ホントすごいよね〜。

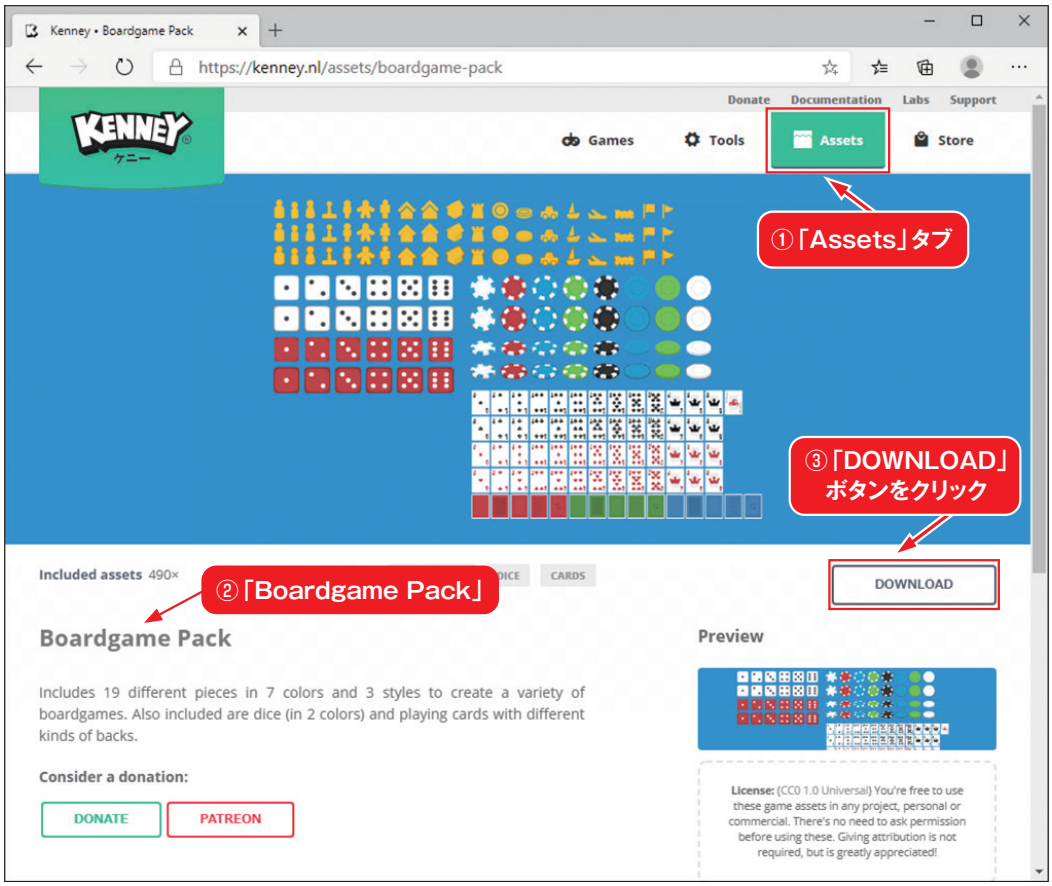
**先生** 「Assets」タブをクリックして「Boardgame Pack」を探そう。または、次のURLからアクセスする。

```
https://kenney.nl/assets/boardgame-pack
```

Boardgame Packは見ての通り、ボードゲームで使えそうなトランプの画像やサイコロの画像の素材集だ。まずは、これを使おう。「DOWNLOAD」ボタンをクリックすると（**図2**）、

「boardgamepack.zip」というファイルをダウンロードできる。

**図2●Boardgame Packをダウンロード**



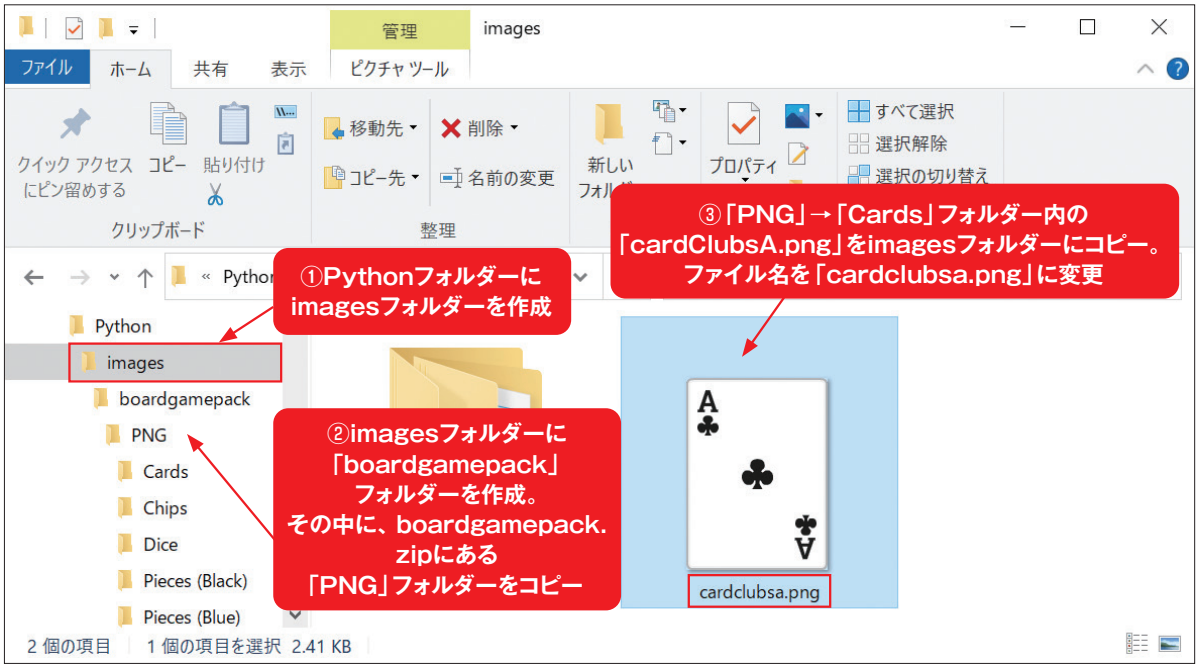

**タツヤ** できた!

**先生** じゃあ、先ほど作ったPythonフォルダーの中に「images」フォルダーを作ろう。さらに、imagesフォルダーの中に「boardgamepack」フォルダーを作る。

**レイ** フォルダーを作る、っと。

**先生** そしたら、boardgamepack.zipの中にある「PNG」フォルダーを、boardgamepackフォルダーにコピーしよう(**図3**)。

**図3●boardgamepackフォルダーを作って、PNGフォルダーをコピー**




**レイ** おお、中にはフォルダーがたくさんある。

**先生** PNGフォルダーをコピーできたら、その中にある「Cards」フォルダーを見てみよう。この中に「cardClubsA.png」という画像ファイルがある。

**タツヤ** トランプの画像だ!

**先生** そう。まずは復習を兼ねて、この画像を表示するプログラムを作ってみよう。cardClubsA.pngをimagesフォルダーの直下にコピーして、ファイル名をすべて小文字の「cardclubsa.png」に変更しておこう。

**レイ** 何で小文字にするんだっけ?

**先生** ファイル名やフォルダー名は、アルファベットの小文字だけを使うのが、今回も利用する「Pygame Zero」のルールなんだ。次は、Pygame Zeroをインストールしよう。

1日目

# Part 3 Pygame Zeroのインストール

**タツヤ** Pygame Zeroって、Pythonのゲーム用ライブラリだったよな?

**レイ** 私、インストールの仕方忘れちゃったわ…。

**先生** 実は、今回導入したPython 3.9.1では、Pygame Zeroのインストールがちょっと面倒になったんだ。前回は、コマンドプロンプトから「pip install pgzero」と入力するだけでインストールできた。ところが、Python 3.9.1では、現時点(2021年2月時点)で、“下準備”が必要なんだ。

**タツヤ** 何と!

**先生** 下準備というのは、Python 3.9.1に対応した「pygame」のインストール作業だ。

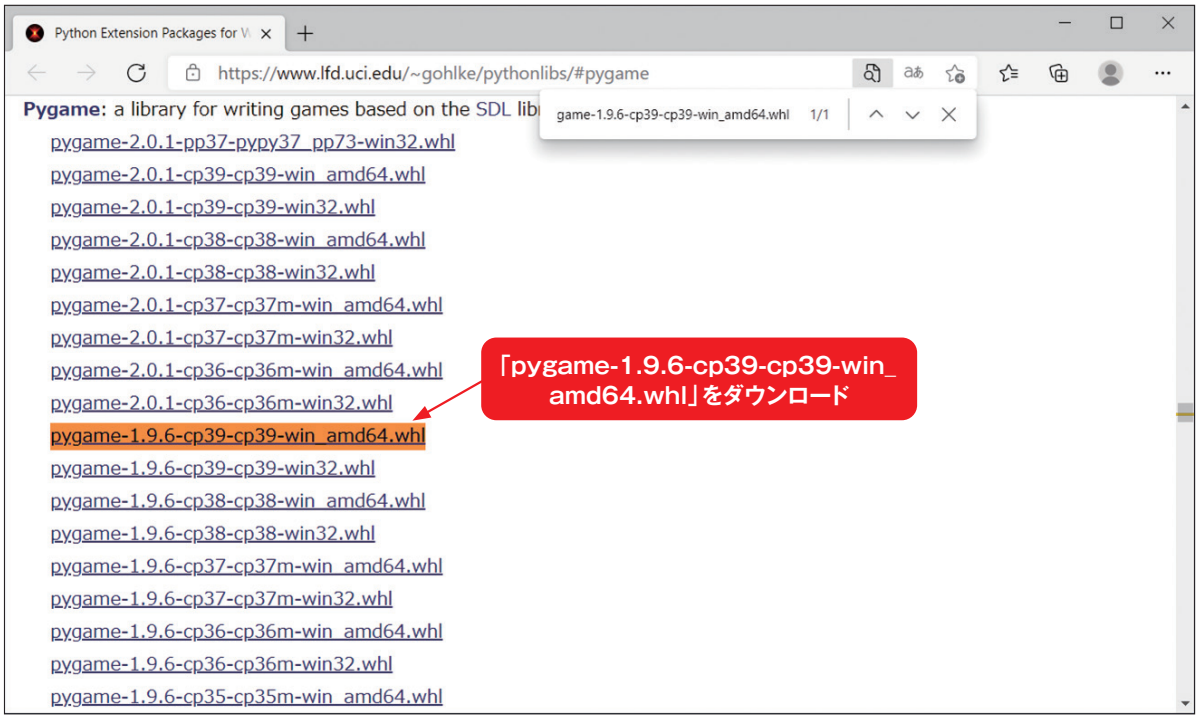
**タツヤ** pygame…?「Pygame Zero」じゃなくて?

**先生** Pygame Zeroは、pygameというゲーム向けライブラリを使いやすくしたものなんだ。今回は、「pygame-1.9.6-cp39-cp39-win_amd64.whl」というファイルをネットで探して、pygameをインストールしよう。例えば、次のURLからダウンロードできる。

```
https://www.lfd.uci.edu/~gohlke/pythonlibs/#pygame
```

似たファイルがたくさんあるから注意しよう(**図1**)。

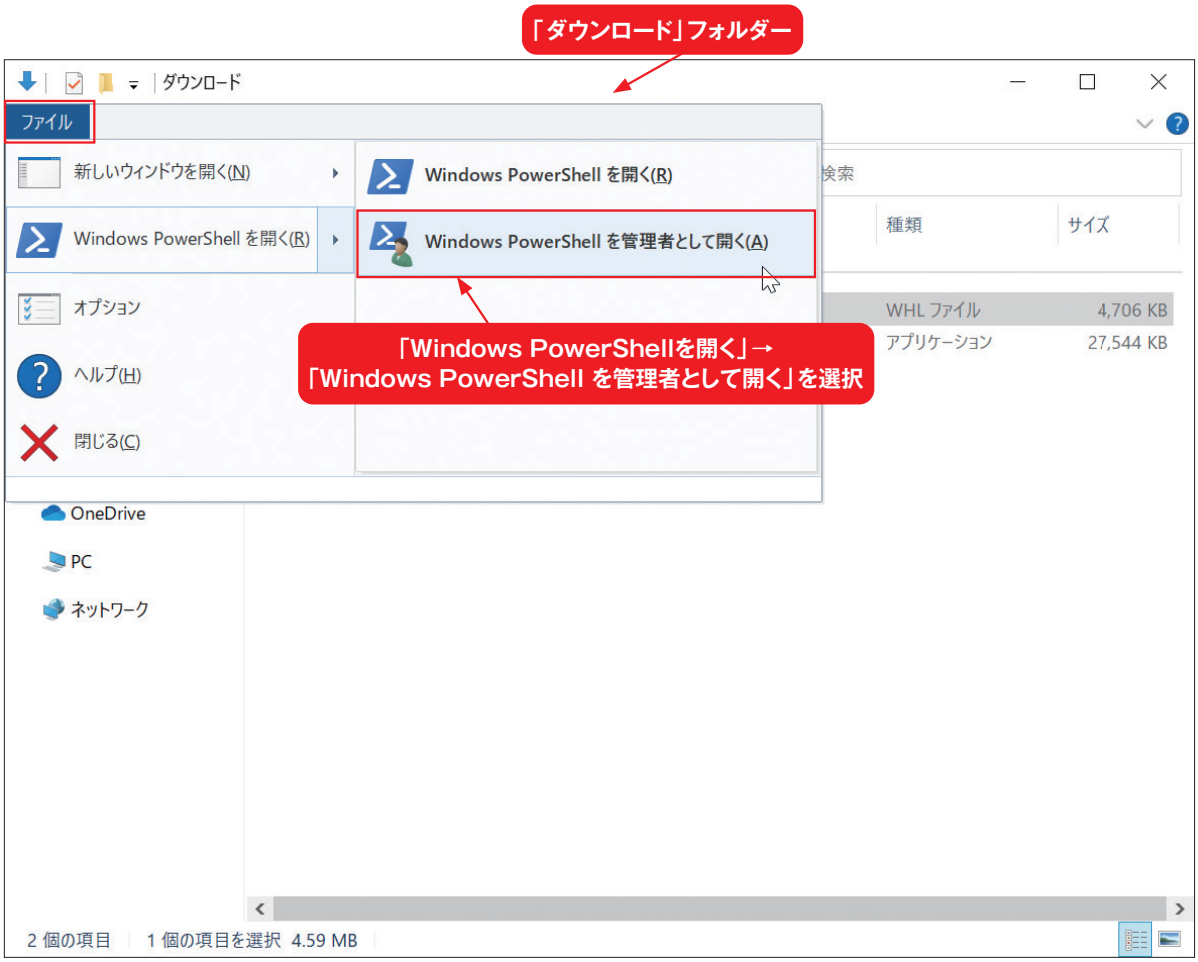
**図1●pygameをダウンロード**




**レイ** ダウンロードしたよ。

**先生** 「ダウンロード」フォルダーにダウンロードできたようだね。それじゃあ、エクスプローラーでダウンロードフォルダーを開いて、「ファイル」メニューの「Windows PowerShellを開く」→「Windows PowerShell を管理者として開く」を選択して、Windows PowerShellを起動しよう（**図2**）。

**図2●Windows PowerShellを起動**




**タツヤ**「pip」コマンドでインストールするの?

**先生**そう。ただ、インストールの前に、pipのバージョンをアップグレードしておこう。「python -m pip install --upgrade pip」のコマンドを入力する。

[管理者：Windows PowerShell]

```
PS C:¥Users¥User¥Downloads> python -m pip install --upgra
de pip
Collecting pip
  Downloading pip-20.3.3-py2.py3-none-any.whl (1.5 MB)
     |████████████████████████████████████████████████████
| 1.5 MB 3.3 MB/s
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 20.2.3
    Uninstalling pip-20.2.3:
      Successfully uninstalled pip-20.2.3
Successfully installed pip-20.3.3
PS C:¥Users¥User¥Downloads>
```

入力

**レイ** アップグレードに成功したっぽい。

**先生** じゃあ、ダウンロードしたpygameをインストールしよう。「pip install .¥pygame-1.9.6-cp39-cp39-win_amd64.whl」と入力する。

[管理者：Windows PowerShell]

```
PS C:¥Users¥User¥Downloads> pip install .¥pygame-1.9.6-
cp39-cp39-win_amd64.whl
Processing c:¥users¥user¥downloads¥pygame-1.9.6-cp39-cp39-
win_amd64.whl
Installing collected packages: pygame
Successfully installed pygame-1.9.6
PS C:¥Users¥User¥Downloads>
```

入力

**タツヤ** とりあえず、うまくいったのかな?

**先生** 「Successfully installed pygame-1.9.6」と表示されていれば、インストールは成功だ。それでは、「pip install pgzero」と入力して、Pygame Zeroをインストールしよう。

[管理者：Windows PowerShell]

```
PS C:¥Users¥User¥Downloads> pip install pgzero
Collecting pgzero
  Downloading pgzero-1.2-py3-none-any.whl (69 kB)
     |████████████████████████████████████████| 69 kB 4.8 MB/s
Requirement already satisfied: pygame<2.0,>=1.9.2 in c:¥users¥user¥appdata¥local¥
programs¥python¥python39¥lib¥site-packages (from pgzero) (1.9.6)
Collecting numpy
  Downloading numpy-1.19.4-cp39-cp39-win_amd64.whl (13.0 MB)
     |████████████████████████████████████████| 13.0 MB 6.8 MB/s
Installing collected packages: numpy, pgzero
Successfully installed numpy-1.19.4 pgzero-1.2
PS C:¥Users¥User¥Downloads>
```

入力

**レイ** ふ〜、終わったわ…。

**先生** 最後に、もう一つやることがある。

**タツヤ** え! まだ何か?

**先生** Pygame Zeroは、内部で「numpy」という計算用のライブラリを使っているので、Pygame Zeroと同時に、numpy 1.19.4がインストールされる。ところが、このnumpy 1.19.4は、現時点ではWindows 10だとエラーになってしまう。

**タツヤ** 何と…。複雑すぎ…。

**先生** これは「ライブラリの依存問題」と言って、複数のライブラリを組み合わせて使う時にしばしば発生する問題なんだ。とりあえず、numpyのバージョンが1.19.4以上だった場合は、1.19.3にダウングレードしておこう。この作業もpipコマンドでできる。

[管理者：Windows PowerShell]

```
PS C:¥Users¥User¥Downloads> pip install numpy==1.19.3
Collecting numpy==1.19.3
  Downloading numpy-1.19.3-cp39-cp39-win_amd64.whl (13.3 M
B)
     |████████████████████████████████████████████████████
| 13.3 MB 6.4 MB/s
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.19.4
    Uninstalling numpy-1.19.4:
      Successfully uninstalled numpy-1.19.4
Successfully installed numpy-1.19.3
PS C:¥Users¥User¥Downloads>
```

入力

**レイ** バージョンを上げたり下げたり、めんどくちゃ〜。

**先生** おつかれさま。これでPygame Zeroのインストールは完了だ。Windows Power Shellは閉じて構わない。次は、少しだけPygame Zeroを復習しておこう。

1日目

## Part 4 Pygame Zeroの基本を復習

**先生** まずは、基本となる、ウインドウを表示するプログラムからだ。

**レイ** 確か、2行くらいで表示できたような気がするわ。

**先生** そうだったね。「IDLE（Python 3.9 64-bit）」を起動して、先ほどの「mygame.py」を、次のコードに書きかえて、実行しよう。

mygame.py

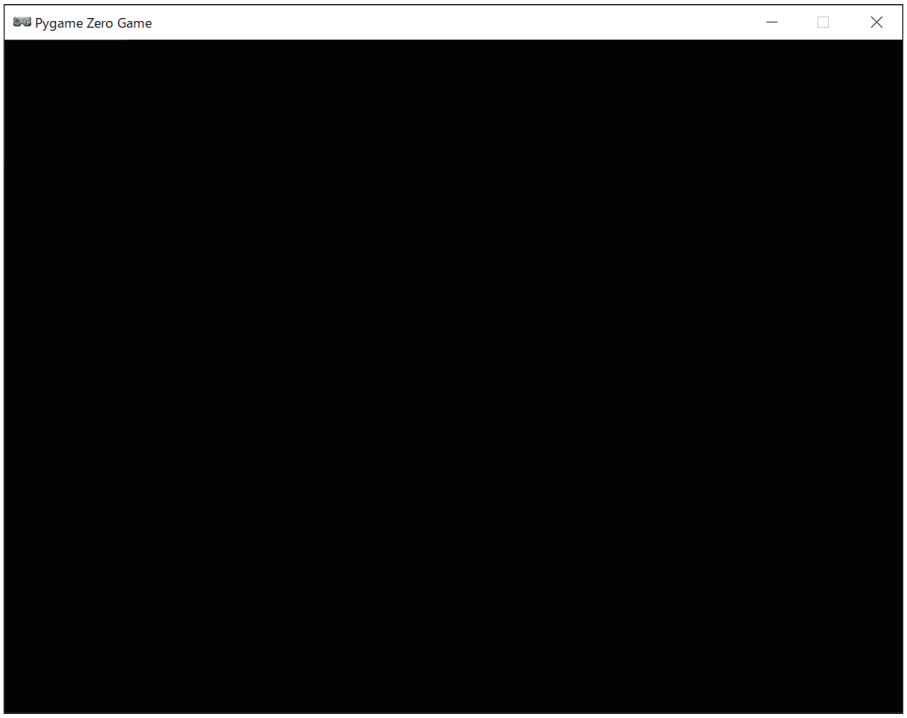
```
import pgzrun

pgzrun.go()
```

Pygame Zeroのオブジェクトを利用できるようにする

Pygame Zeroのプログラムをスタートする

**タツヤ** 出た出た。真っ黒なウインドウが出たよ（**図1**）。

**図1●Pygame Zeroを使って表示したウインドウ**




**先生** プログラムの動作を確認できたらウインドウは閉じておこう。「import pgzrun」のコードは、pgzrunというオブジェクトをプログラムから利用するために記述している。次の「pgzrun.go()」は、pgzrunオブジェクトのgoメソッドを実行する、という意味。これでプログラムがスタートする。では、次のコードに書き換えてみよう。

**mygame.py**

```
import pgzrun

WIDTH = 200
HEIGHT = 200
card = Actor('cardclubsa', center=(100,100))

def draw():
    screen.clear()
    card.draw()

pgzrun.go()
```

ウインドウの幅と高さ

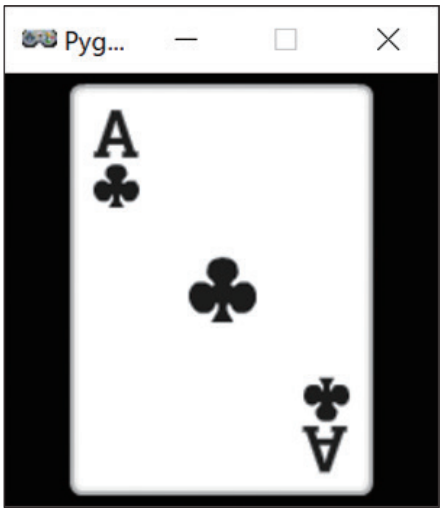
画像ファイル名

画像の中央をウインドウのx座標100、y座標100の位置に配置

ウインドウのクリアとActorオブジェクトの描画

**レイ** 書き換えたから、実行してみるね（**図2**）。

**図2●トランプのカードを表示**




**タツヤ** あれ? エイリアンの画像が出てこない。

**レイ** 当たり前じゃん。imagesフォルダーにコピーしたのはトランプの画像だったじゃない。

**先生** プログラムで利用する画像ファイルはimagesフォルダーに格納するのがPygame Zeroのルールだったね。このimagesフォルダーはプログラムのファイルと同じフォルダーに置いておかなければならない。

**タツヤ** そうでした!

**先生** で、このコードでは、「WIDTH」と「HEIGHT」で、表示するウインドウの横幅と高さを決めている。

**タツヤ** これらは変数なんだね。

**先生** そう。で、「Actor」は画像を表示するために必要なオブジェクトだ。Actor関数を実行することで、画像ファイルのcardclubsa.pngを読み込んでActorオブジェクトを作り、そのActorオブジェクトと変数cardを、「=」演算子で結び付ける。

**レイ** centerって引数は、何の真ん中?

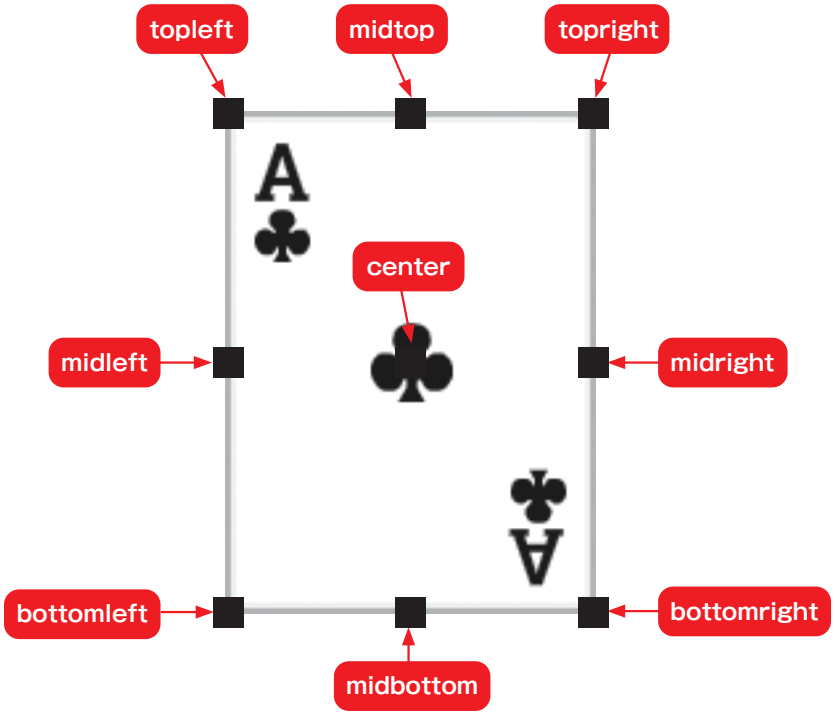
**先生** 「center=(100,100)」は、Actorの中央座標を、ウインドウのx座標100、y座標100の位置にする、という意味。

**レイ** 丸括弧で括っているから、(100,100)は「タプル」だよね。

**タツヤ** タプルって、要素を変更できない「リスト」だよね?

**先生** その通り。よく覚えていたね。Actorオブジェクトには、それぞれ名前が付いた座標点がある(**図3**)。centerとかmidtopとかね。で、それらの座標点に、タプルを使ってActorの位置を設定するんだ。

**図3●Actorオブジェクトの座標点**




**レイ** 「def」は、自分で関数を作るときに使うのよね。

**先生** レイちゃんも、思い出してきたね。ここでは「draw」という名前の関数を作っている。draw関数はその名前の通り、描画するための関数だ。ここでは、ウインドウの中身を

クリアして、トランプの画像を描画している。

**タツヤ** そうそう。いろいろ思い出してきた!

**先生** それでは、さらにコードを追加して、トランプの画像を移動させよう。次のコードを実行してみよう。

mygame.py

```
import pgzrun

WIDTH = 200
HEIGHT = 200
card = Actor('cardclubsa', midright=(0,100))

def draw():
    screen.clear()
    card.draw()

def update():
    card.x += 1
    # card.left += 1

pgzrun.go()
```

1秒間に60回呼び出されるupdate関数

Actorのx座標を増やしてゆくと右へ移動する

**タツヤ** よし、実行(図4)。

**図4●Actorオブジェクトの移動**

**先生** Pygame Zeroは、1秒間に60回、「update関数」を実行することを覚えているかな? このupdate関数は、さらにdraw関数を内部で実行する。そこで、update関数にActorオブジェクトの座標を変化させるコードを書くと、Actorオブジェクトが移動するんだ。このとき、Acterオブジェクトの「x」属性を使ってもいいし、「left」属性を使ってもいい。属性は、オブジェクトに属する変数や関数のことだね。

**レイ** 結構覚えてて、自分でもビックリだわ。

**先生** 今日はこれくらいにしよう。この続・Pythonゲームコースはゲームの作り方の説明を主眼にしているから、Pythonの基本文法にはあまり触れません。だから、Pythonの文法の予習をしっかりしておくこと。お疲れ様。オンラインをオフにします。また明日。

**レイ** **タツヤ** バイバ〜イ。

2日目

# 神経衰弱ゲームを作ろう

# 2日目 Part 1 カードの表示

**先生** 今日は、神経衰弱のゲームを作ろう。ただ、プログラムが複雑にならないように、ちょっと簡単な神経衰弱にする。具体的には、8枚のカードを使って、1人でプレイできるゲームだ。

**タツヤ** オーケー! 基本さえ教えてもらえれば、この「タツヤ」がガンガンアップデートいたしますよ〜。

**レイ** なんか、気合い入りまくり〜。

**先生** まずは、神経衰弱で利用するカードの画像を用意しよう。Pythonフォルダーの下に作ったimagesフォルダー→boardgamepackフォルダー→PNGフォルダー→Cardsフォルダーに、トランプの画像が一式ある。この中から、次の画像ファイルを探して、imagesフォルダーにコピーする。

```
cardClubsA.png
cardClubs2.png
cardClubs3.png
cardClubs4.png
cardHeartsA.png
cardHearts2.png
cardHearts3.png
cardHearts4.png
cardBack_blue5.png
```

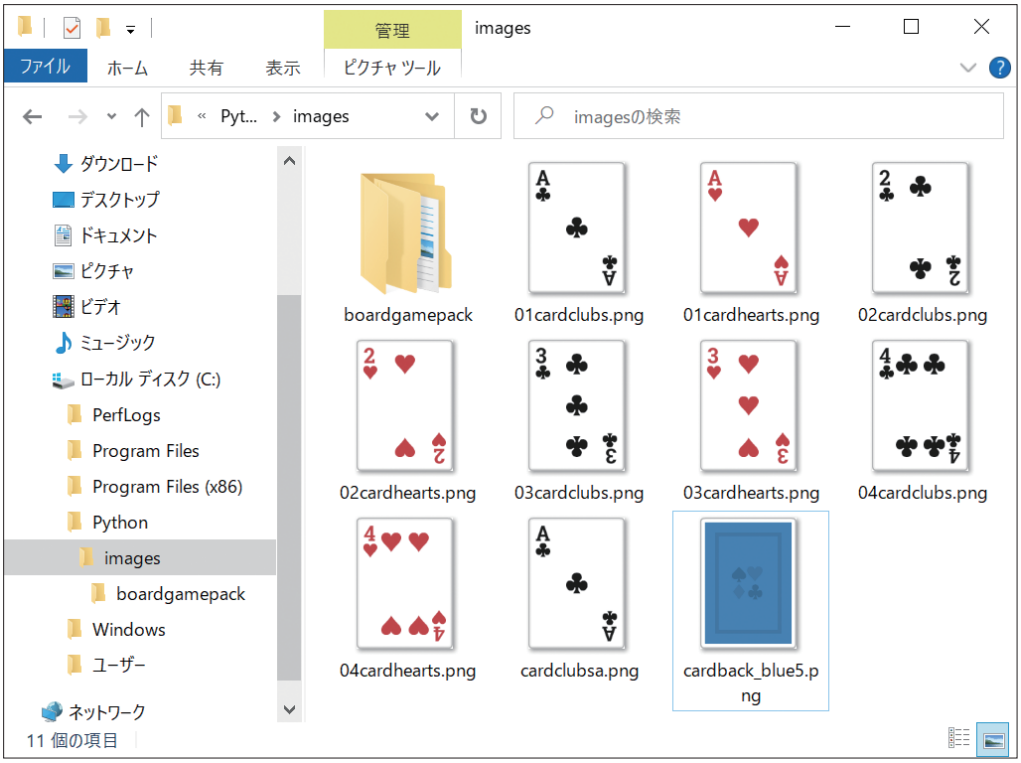
**タツヤ** クラブとハートのカードのAから4を使うんだね。

**レイ** トランプのカードって、どっちが表でどっちが裏なの?

**先生** トランプは数字や絵柄が印刷されている方が表で、模様が印刷されている方が裏だ。画像ファイルのコピーが終わったら、カードのファイル名をすべて、次のように変更する(**図1**)。

```
cardClubsA.png   → 01cardclubs.png
cardClubs2.png   → 02cardclubs.png
cardClubs3.png   → 03cardclubs.png
cardClubs4.png   → 04cardclubs.png
cardHeartsA.png  → 01cardhearts.png
cardHearts2.png  → 02cardhearts.png
cardHearts3.png  → 03cardhearts.png
cardHearts4.png  → 04cardhearts.png
cardBack_blue5.png → cardback_blue5.png
```

**図1●神経衰弱のプログラムで利用する画像ファイル**




**タツヤ** 小文字にするのはPygame Zeroのルールだからわかるとして、何で「01」とか「02」とか付けるの?

**先生** これは、ファイル名の先頭2文字を調べれば、そのカードが数字のいくつなのかを判別できるようにするためだ。

**レイ** コンピュータは、絵柄を判別できないもんね〜。

**タツヤ** そんなことないぞ〜。AIならわかるぞ〜。

**先生** 確かにそうだ。ただ、できるだけ簡単に判別できるように、今回はファイル名を利用しよう。

**タツヤ** まあ、それが普通だよね。

**先生** 次は、このトランプのカードを表示するウインドウを考える。クラブとハートのカードを4×2で8枚並べたい。

**レイ** カードのサイズは、140×190ピクセルだわ。画像のサイズを変更した方がいいのかしら?

**先生** 今回は面倒なのでこのまま使おう。カード同士の間隔は10ピクセルとする。

**タツヤ** つまり、横610、縦410のウインドウを作るのか…。

**レイ** てか、計算…ハヤ!

**タツヤ** へへっ、横は10+140+10+140+10+140+10+140+10、縦は10+190+10+190+10だからね。

**先生** さすがタツヤ君。それじゃ、IDLEを起動して、「New　File」でファイルを作ったら、いったん「memory.py」の名前でPythonフォルダーに保存する。で、カードの裏面が4×2で並んでいるプログラムを作っ…。

**レイ** できました!

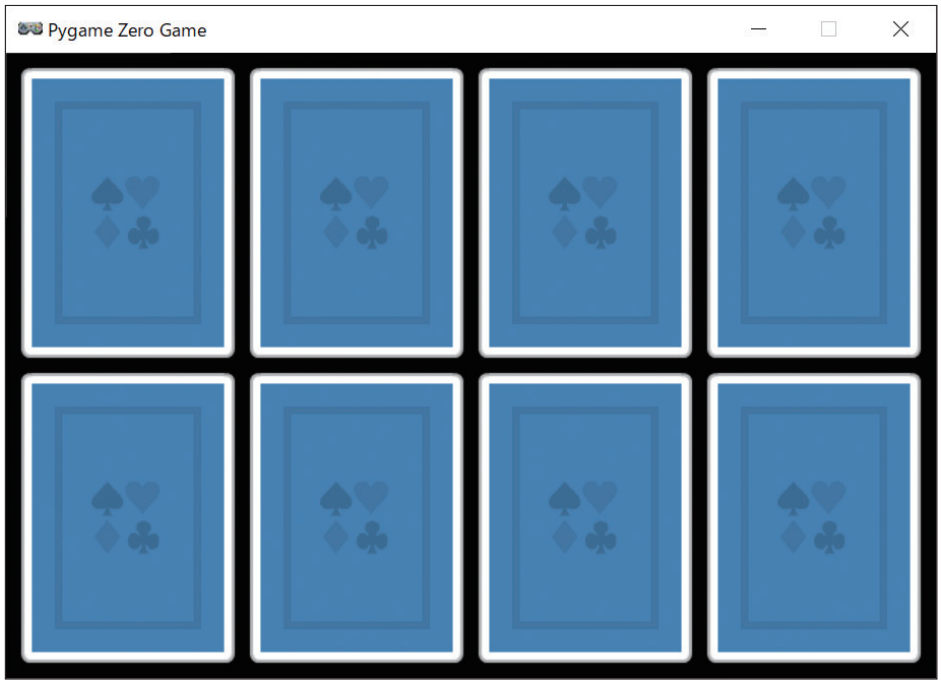
memory.py

```
import pgzrun

WIDTH = 610 # ウインドウの横幅
HEIGHT = 410 # ウインドウの縦幅
card_w = 140 # カードの横幅
card_h = 190 # カードの縦幅

card_b = 'cardback_blue5' # カードの裏
card_c1 = Actor(card_b, topleft=(10,10)) # カードのActor
card_c2 = Actor(card_b, topleft=(160,10))
card_c3 = Actor(card_b, topleft=(310,10))
card_c4 = Actor(card_b, topleft=(460,10))
card_h1 = Actor(card_b, topleft=(10,210))
card_h2 = Actor(card_b, topleft=(160,210))
card_h3 = Actor(card_b, topleft=(310,210))
card_h4 = Actor(card_b, topleft=(460,210))

# ウインドウ内の描画
def draw():
    screen.clear()
    card_c1.draw()
    card_c2.draw()
    card_c3.draw()
    card_c4.draw()
    card_h1.draw()
    card_h2.draw()
    card_h3.draw()
    card_h4.draw()

pgzrun.go()
```

タツヤ マジ早…。

レイ 実行するよ（図2）。

**図2●memory.pyの実行結果**




**先生** おっ、おおぉ、すばらしい。キレイに表示できたね。ただ…。

**レイ** ただ…?

**先生** レイちゃんのプログラムは、とても良くできているんだが…。ちょっと、困ったことがある。

**レイ** え〜、うそ〜、なに〜?

**先生** このプログラムでは、カードの位置を直接数値で入力している。そうすると、今後、例えば間隔を変更したくなったときに、とても面倒だ。

**タツヤ** そうだ! 変数と繰り返しを使うんだ!

**先生** その通り。そこで、こんな風に書き換えてみたので参考にして欲しい。

memory.py

```
import pgzrun

WIDTH = 610    # ウインドウの横幅
HEIGHT = 410   # ウインドウの縦幅
card_w = 140   # カードの横幅
card_h = 190   # カードの縦幅

card_b = 'cardback_blue5'   # カードの裏

card = []   # カードのActorのリスト
x = 10      # カード配置用のx座標
y = 10      # カード配置用のy座標
m = 10      # カード同士の間隔
w = 4       # 横の数
h = 2       # 縦の数

# カードの生成と配置座標のセット
for i in range(h):
    for n in range(w):
        # カードのActorを生成
        card.append(Actor(card_b, topleft=(x,y)))
        x += card_w + m
    x = m
    y += card_h + m

# ウインドウ内の描画
def draw():
    screen.clear()
    for c in card:
        c.draw()

pgzrun.go()
```

**タツヤ** なるほど。カードはPythonのリストに格納するんだ。

**レイ** ふ〜ん。リストなら繰り返しは余裕ね。for文で簡単に処理できるわ。繰り返す回数を意識する必要もない…。座標は直接じゃなくて計算で求めてる…。完敗ね…。

**先生** いやいや、レイちゃんのコードが悪いわけじゃない。ただ、変数、リスト、繰り返し

のテクニックを使えば、より変更に強いプログラムになるってことだ。

# 2日目 Part 2 カードをめくる処理

**タツヤ** カードは表示できたけど、どうやってめくるんだ?

**レイ** マウスでカードをクリックしたら、そのカードを表にすればいいんじゃない?

**タツヤ** でも、どうやって…。

**先生** 私も、レイちゃんと同じ考えだ。マウスでクリックして選択する方法がいいだろう。

**タツヤ** キーが押されたら呼び出される関数があったけど、マウスにも同じようなのがあるんだっけ?

**先生** もちろん、ある。マウスやキーのイベントの解説は、Pygame Zeroのマニュアルに詳しく書いてあるので、もう一度見直しておくこと(**図1**)。

**図1●Pygame Zeroのマニュアル**
**https://pygame-zero.readthedocs.io/ja/latest/index.html**

**レイ** ふむふむ。

**先生** マニュアルの「イベントのフック」→「イベント処理のフック」を読もう。ウインドウ内でマウスをクリックすると呼び出される関数として、「on_mouse_down(pos, button)」というイベントハンドラの関数が紹介されている。引数posに渡されるデータは、クリックしたときのマウスカーソルの座標だ。引数buttonに渡されるデータは、クリックしたマウスボタンの種類だ。

**レイ** でも、どのカードをクリックしたのかって、どうやって調べるの?

**先生** それには、Actorオブジェクトが持っている「collidepoint」メソッドを使う(**表1**)。

**表1●Actorオブジェクトが持つ衝突判定メソッド**

| 衝突判定メソッド | 使い方 |
|---|---|
| Actor オブジェクト .colliderect(rect) | 引数は Rect オブジェクト（Rect オブジェクトは Rect 関数で生成する。Rect 関数の第1引数は、左上のx座標とy座標のタプル。第2引数は幅と高さのタプル）。接触しているとTrue を返す。 |
| Actor オブジェクト .collidepoint(pos) | 引数は pos（x 座標とy 座標のタプル）。接触しているとTrue を返す。 |

**タツヤ** このメソッド、前回作ったシューティングゲームで使ったメソッドの仲間だ。マウスがクリックされたら、カードのオブジェクトのcollidepointメソッドで、マウスカーソルの座標とカードが衝突しているか、調べればいいんだ。

**先生** そう。では、カードをクリックしたら、クラブAの画像に切り替わるようにするon_mouse_down関数を紹介しよう。

memory.py

```
import pgzrun

WIDTH = 610 # ウインドウの横幅
HEIGHT = 410 # ウインドウの縦幅
card_w = 140 # カードの横幅
card_h = 190 # カードの縦幅

card_b = 'cardback_blue5' # カードの裏

card = [] # カードのActor
x = 10 # カード配置用のx座標
y = 10 # カード配置用のy座標
m = 10 # カード同士の間隔
w = 4  # 横の数
h = 2  # 縦の数

# カードの生成と配置座標のセット
for i in range(h):
    for n in range(w):
        # カードのActorを生成
        card.append(Actor(card_b, topleft=(x,y)))
        x += card_w + m
    x = m
    y += card_h + m

# ウインドウ内の描画
def draw():
    screen.clear()
    for c in card:
        c.draw()

# マウスクリックの処理
def on_mouse_down(pos):
    for c in card:
        if c.collidepoint(pos):
            c.image = '01cardclubs' # カードをクラブのAにする

pgzrun.go()
```

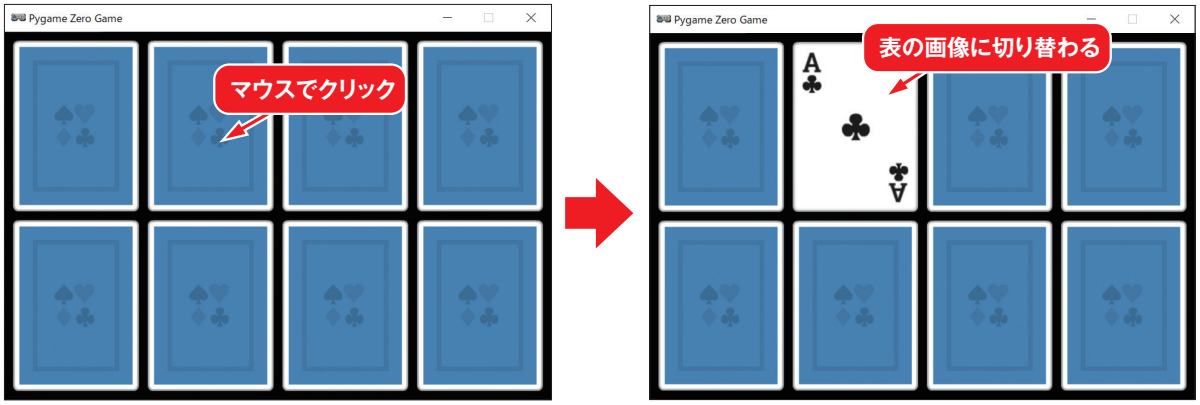
マウスカーソルの座標とカードの座標の衝突判定

座標が重なっていたら画像を変更

レイ マウスでクリックするとon_mouse_down関数が呼ばれるのね。そして、マウスカーソルがカード内ならcollidepointメソッドがTrueを返すから、そのカードの画像をクラ

ブのAに変更している（**図2**）。

**図2●クリックすると表になる**




**先生** ただ、このままでは画像が切り替わるだけだ。神経衰弱にするには、1枚目と2枚目を分けて管理する必要がある。

**タツヤ** つまり、2枚目をめくったら数字が一致しているか判断して、一致していたらそのまま、違っていたらカードを元に戻す…。こりゃ、面倒だぁ。

**先生** さらに、2枚目をめくって一致していなかったら、すぐに裏に戻すのではなく、2枚目のカードを少し見せてから元に戻さないと、プレイヤーがカードを覚える時間がない。

**レイ** 2枚目が見えないと、神経衰弱にならないものね。少し時間がたってから処理をするって、どうやるのかしら? Scratchみたいに「○秒まつ」ブロックを使うの?

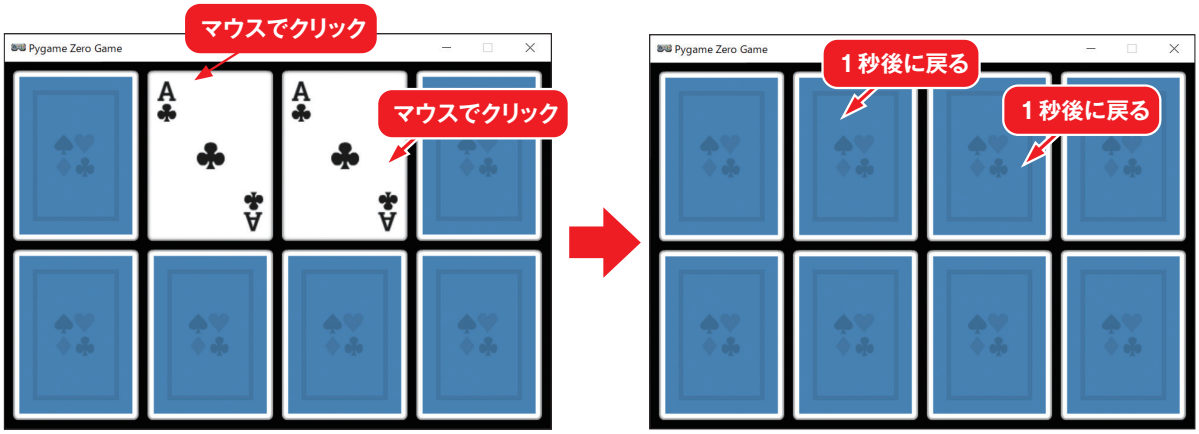
**先生** Pythonにも「○秒まつ」ブロックと同じ処理を行えるtime.sleep関数があるけど、この関数は全体の処理を停止してしまうので、使い勝手が良くない。ここでは、Pygame Zeroが持っているclockオブジェクトの「schedule_unique」メソッドを使おう。

**レイ** マニュアルにも書いてあるわ。オリジナルの関数を用意しておいて、その関数を遅れて実行させることができるみたい。

**タツヤ** 便利じゃん!

**先生** それでは、実習タイムだ。先ほどのon_mouse_down関数を参考に、2枚のカードをめくったら、1秒後に裏に戻るというコードを書いて欲しい（**図3**）。

**図3●1秒後にカードの裏に戻る**




・・・

**レイ** う〜ん、エラーにはならないけど動かない…。

**タツヤ** できたっポイ…。あっ、ダメだ。待っている間に別のカードをクリックできちゃう…。難しい…。

**先生** 関数内で値を変更するグローバル変数は、関数内でglobal宣言することがポイント。schedule_uniqueメソッドで関数が呼び出されるのを待っている間は、マウスクリックを無効化する処理も必要だね。それじゃ、次のコードを参考に考えてみて。

**memory.py**

```
import pgzrun
・・・（略）・・・

count = 0 # めくった順番
turn_card = [] # めくったカードの記録用リスト
wait_flg = False # 待ち時間中はTrue

# カードの生成と配置座標のセット
・・・（略）・・・
```

## memory.pyの続き

```
# ウインドウ内の描画
・・・(略)・・・

# マウスクリックの処理
def on_mouse_down(pos):
    global count
    global turn_card
    global wait_flg
    if wait_flg: # 1秒間、マウスクリックを無効化
        return
    for c in card:
        if c.collidepoint(pos):
            c.image = '01cardclubs' # カードをクラブのAにする
            turn_card.append(c) # めくったカードを記録
            count += 1
    if count == 2: # 2枚目をめくった
        count = 0
        # 1秒後にrestore関数を呼びだす
        clock.schedule_unique(restore, 1)
        wait_flg = True # マウスクリックを無効化

# 1秒後にカードを裏に戻す関数
def restore():
    global turn_card
    global wait_flg
    turn_card[0].image = card_b # カードを裏に戻す
    turn_card[1].image = card_b # カードを裏に戻す
    turn_card.clear()
    wait_flg = False

pgzrun.go()
```

第2引数は待つ秒数

マウスクリックを有効に

**レイ** そうかぁ、関数内で値を変更するグローバル変数は、globalで宣言しないとダメだったわね。

**タツヤ** なるほど。表を表示しているカードをグルーバルなリストのturn_cardに入れておけば、別の関数でそのカードだけ画像を変更できるのか…。

**先生** マウスカーソルのイベントを無効にする処理は、自分でフラグを使って処理すると簡単だ。それじゃ、いよいよ。神経衰弱ゲームとして仕上げて行こう。

# 2日目 Part 3 ゲームクリアで正解率を表示

**先生** さぁ、基本的な処理はできてきた。最後にゲームとして仕上げていこう。

**タツヤ** よっしゃ〜!

**先生** まずは、今はどれをクリックしても同じカードなので、異なる数字のカードになるように各画像のファイル名をリストに格納しておく。そして、リスト内のカードがランダムに並ぶように、randomオブジェクトの「shuffle」メソッドを使って、シャッフルしておこう。

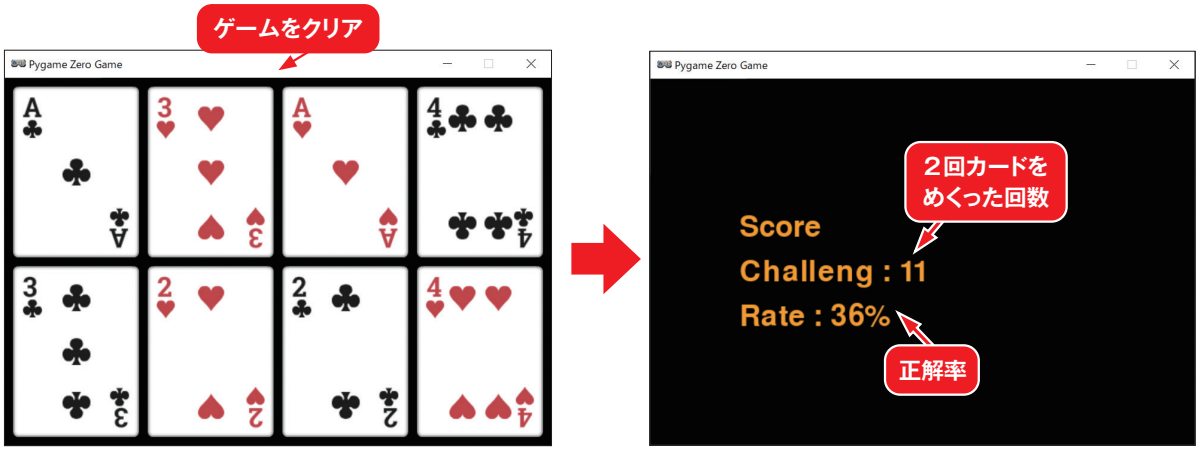
```
# カードの表のリスト生成とシャッフル
card_f = []
for i in range(1,5):
    card_f.append('{:02}'.format(i) + 'cardclubs')
    card_f.append('{:02}'.format(i) + 'cardhearts')
random.shuffle(card_f)

# カードの裏
・・・(略)・・・
```

**タツヤ** format関数でファイル名の先頭2文字を作り出している…。シビレルなぁ。Scratchだとこんなコード、1行で書けないなぁ。

**レイ** シャッフルするメソッドがあるのね。便利〜。

**先生** ところで、記憶力や運が強い人は、正解率が高くなる。そこで、2回カードをめくった回数を変数totalに記録する。最後に、「total ÷ 一致したセット数」を正解率として、ゲームクリア後に表示したい(**図1**)。

図1●ゲームクリア後に正解率を表示




**タツヤ** 正解率の表示は、draw関数の中で「screen.draw.text関数」を使うんだよね!

**レイ** スゴ〜イ…、と思ったけど、前回、やったよねソレ。

**先生** 最初に、マウスでクリックしたカードが裏だったら表にして、2回目のクリックなら数字が同じかどうかを判断するコードの例を紹介しよう。

memory.py

```
# カードの裏
・・・(略)・・・

count = 0 # めくった順番
total = 0 # 2回めくった回数
point = 0 # 数字がそろうと1ポイント

・・・(略)・・・

# マウスクリックの処理
def on_mouse_down(pos):
    global count
    global turn_card
    global wait_flg
    if wait_flg:
        return
    for i in range(len(card)):
        if card[i].collidepoint(pos):
            if card[i].image == card_b:
```

カードが裏かどうかの判断

### memory.pyの続き

```
                card[i].image = card_f[i] # カードを表にする
                turn_card.append(card[i]) # めくったカードを記録
                count += 1
    if count == 2:
        count = 0
        clock.schedule_unique(restore, 1)
        wait_flg = True

# カードの数字が同じか判断して、違うときは裏に戻す関数
def restore():
    global point
    global turn_card
    global wait_flg
    global total
    # 違う数字ならカードを裏返す
    if turn_card[0].image[:2] != turn_card[1].image[:2]:
        turn_card[0].image = card_b
        turn_card[1].image = card_b
    else:
        point += 1 # 同じ数字の時は戻さずにポイントを加算
    total += 1 # 2回めくった回数を増やす
    turn_card.clear()
    wait_flg = False
```

**レイ** めくったカードが同じ数字なら変数pointの値を1増やすのね。

**先生** 今回は数字が同じなら絵札が違ってもよいルールにしたよ。これで、大体の処理は完成だ。では、演習時間としよう。神経衰弱ゲームを完成させて欲しい。

. . .

**レイ** 結構、それっぽくなってきたわ。

**タツヤ** こっちもできてきた。

**先生** それじゃ、終了時間も近づいてきたので、全体のコードを紹介しておこう。

memory.py

```
import pgzrun
import time
import random
import math

WIDTH = 610 # ウインドウの横幅
HEIGHT = 410 # ウインドウの縦幅
card_w = 140 # カードの横幅
card_h = 190 # カードの縦幅

# カードの表のリスト生成とシャッフル
card_f = []
for i in range(1,5):
    card_f.append('{:02}'.format(i) + 'cardclubs')
    card_f.append('{:02}'.format(i) + 'cardhearts')
random.shuffle(card_f)

# カードの裏
card_b = 'cardback_blue5'

card = [] # カードのActor
count = 0 # めくった順番
total = 0 # 2回めくった回数
turn_card = [] # めくったカード
wait_flg = False # 待ち時間中はTrue
point = 0 # 数字がそろうと1ポイント

x = 10 # カード配置用のx座標
y = 10 # カード配置用のyx座標

# カードの生成と配置座標のセット
for i in range(2):
    for n in range(4):
        # カードのActorを生成
        card.append(Actor(card_b, topleft=(x, y)))
        x += card_w + 10
    x = 10
    y += card_h + 10

# ウインドウ内の描画
def draw():
    screen.clear()
```

## memory.pyの続き

```
    if point == len(card)/2: # すべてのカードが一致したらスコア表示
        message = ['Score', 'Challeng : ' + str(total),
'Rate : ' + ¥
            str(math.floor(point/total*100)) + '%']
        for i in range(3):
            screen.draw.text(message[i], (100, 150+i*50),
 ¥
                color='orange', fontsize=48)
    else:
        for c in card:
            c.draw()

# マウスクリックの処理
def on_mouse_down(pos):
    global count
    global turn_card
    global wait_flg
    if wait_flg: # 1秒間、マウスクリックは無効
        return
    for i in range(len(card)):
        if card[i].collidepoint(pos):
            if card[i].image == card_b:
                card[i].image = card_f[i] # カードを表にする
                turn_card.append(card[i]) # めくったカードを記録
                count += 1
    if count == 2: # 2枚目をめくった
        count = 0
         clock.schedule_unique(restore, 1) # 1秒後にrestore
関数を呼びだす
        wait_flg = True # マウスクリックを無効化

# カードの数字が同じか判断して、違うときは裏に戻す関数
def restore():
    global point
    global turn_card
    global wait_flg
    global total
    # 違う数字ならカードを裏返す
    if turn_card[0].image[:2] != turn_card[1].image[:2]:
        turn_card[0].image = card_b
        turn_card[1].image = card_b
    else:
```

### memory.pyの続き

```
            point += 1 # 同じ数字なら裏返さずにポイントを加算
        total += 1 # 2回めくった回数を増やす
        turn_card.clear()
        wait_flg = False

pgzrun.go()
```

**レイ** スコアの表示では、表示する文字列をリストに格納しているのね。こうすれば繰り返しの処理で表示できるのか〜。

**先生** まぁ、細かなテクニック的な部分は経験がものを言うかもね。大切なのは、「読みやすいコードを自分で試行錯誤しながら書く」ってことだ。

**タツヤ** オ〜シ。面白くなってきた。それじゃ、僕は2人で対戦できるように改造しよう。

**レイ** 私は、カードが「ペラッ」ってめくれるアニメーションを追加したいな。

**先生** それでは、今日はここまで。お疲れ様。

3日目

# スロットゲームを作ろう

# 3日目 Part 1 ゲームに必要な画像を用意する

**先生** 今日作るのは、スロットゲームだ。

**タツヤ** あ〜、よく見るやつ。

**レイ** 7が3つそろったら、ポイント7倍とかね。

**先生** スロットゲームは操作が単純だし、誰でも気軽にできるので人気が高い。ただ、リアルに動作するスロットマシンを作ろうとすると、結構な工夫が必要になる。

**タツヤ** 3Dとか?

**先生** 今どきは3Dのゲームエンジンを使えば、本物と見分けがつかないようなリアルなスロットマシンを作ることもできる。

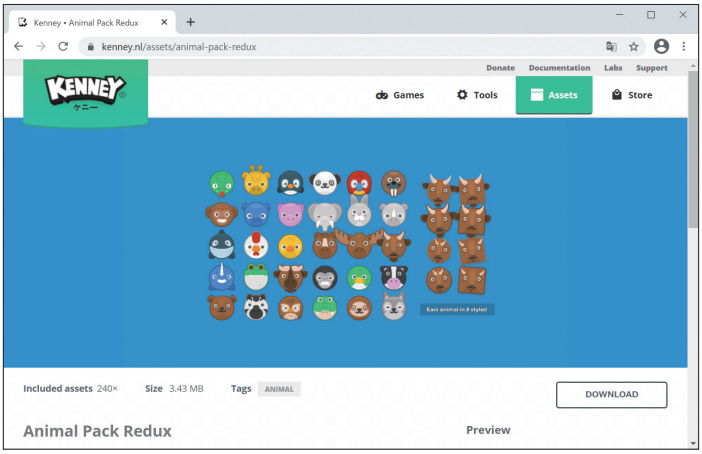
**タツヤ** でも、大変そうだよね。

**先生** その通り。今回はPygame Zeroでお手軽に作ることにしよう。

**タツヤ** お手軽が一番だね。スロットの画像もケニーのサイトからダウンロードするのかな?

**先生** スロットゲーム専用と言える画像は見当たらないので、スロットの絵柄には「Animal Pack Redux」の動物アイコンを使うことにする(**図1**)。

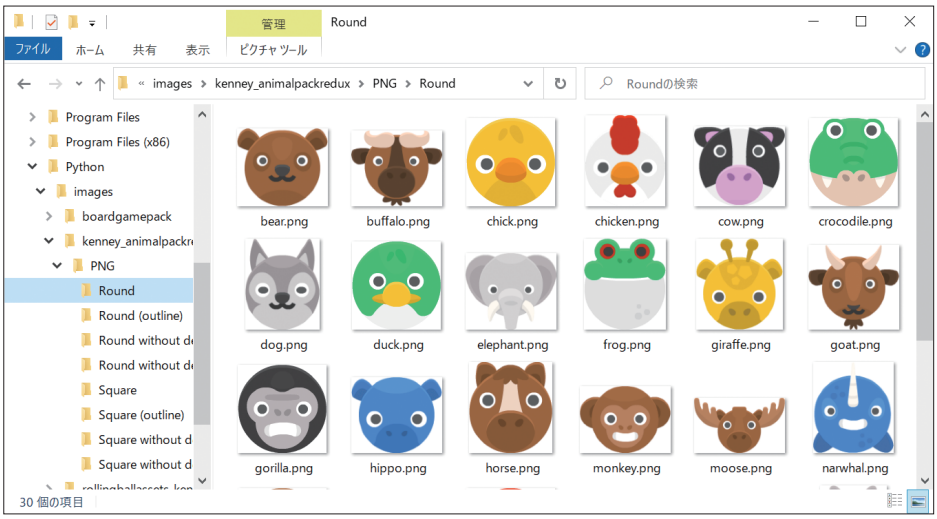
**図1●Animal Pack Reduxの画像をスロットの絵柄に使う。URLはhttps://kenney.nl/assets/animal-pack-redux**



**レイ** かわいい〜。

**先生** 「kenney_animalpackredux.zip」というファイルをダウンロードできる。ダウンロードしたら、imagesフォルダーに「kenney_animalpackredux」フォルダーを作って、kenney_animalpackredux.zipの中にあるPNGフォルダーをコピーしよう（**図2**）。

**図2●Animal Pack Reduxの画像**




**レイ** キャ〜、動物の画像がいっぱい。

**先生** スロットの絵柄として7枚の画像を使おう。PNGフォルダーの中のRoundフォルダーから好きな動物を選んで、imagesフォルダーの直下にコピーする。今回は、次の7枚にしよう。

```
ニワトリ（chicken.png）
牛（cow.png）
キリン（giraffe.png）
ヤギ（goat.png）
馬（horse.png）
フクロウ（owl.png）
パンダ（panda.png）
```

さらに、ファイル名を次のように変更する。

```
chicken.png → slot01.png
cow.png     → slot02.png
giraffe.png → slot03.png
goat.png    → slot04.png
```

```
horse.png   → slot05.png
owl.png     → slot06.png
panda.png   → slot07.png
```

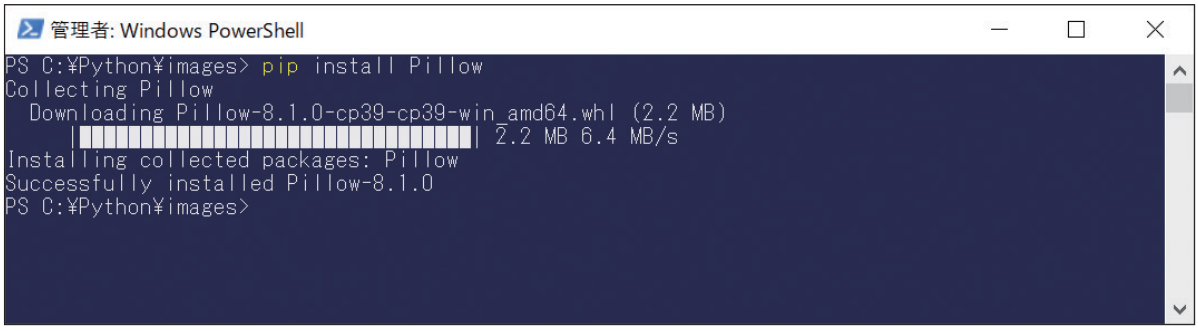
**タツヤ** う〜ん。でも、この画像ってサイズがバラバラだなぁ。

**先生** サイズがそろっていた方がプログラムしやすいので、前回利用したサイズ変更プログラムで、100×100ピクセルにリサイズしておこう。

**レイ** 確か、画像を扱う専用のライブラリが必要だったわよね。

**先生** そう。「Pillow」ライブラリだ。インストールするには、pipを使う。Windows PowerShellを起動して「pip install Pillow」と入力しよう（**図3**）。

**図3●Pillowライブラリをインストール**




**タツヤ** ライブラリのインストールは簡単。

**先生** 次は、サイズ変更プログラムを入力する。IDLEを起動して「File」メニューの「New File」を選択。エディターが起動したら、次のコードを入力しよう。入力が完了したら、Pythonフォルダーに、「img_resize.py」の名前で保存しよう。

**img_resize.py**

```
from PIL import Image

for i in range(7):
    img = Image.open('./images/slot0' + str(i + 1) + '.png')
    img_resize = img.resize((100, 100))
    img_resize.save('./images/slot0' + str(i + 1) + '.png')
```

Imageクラスを使うためのimport文

openメソッドで画像ファイルを読み込む

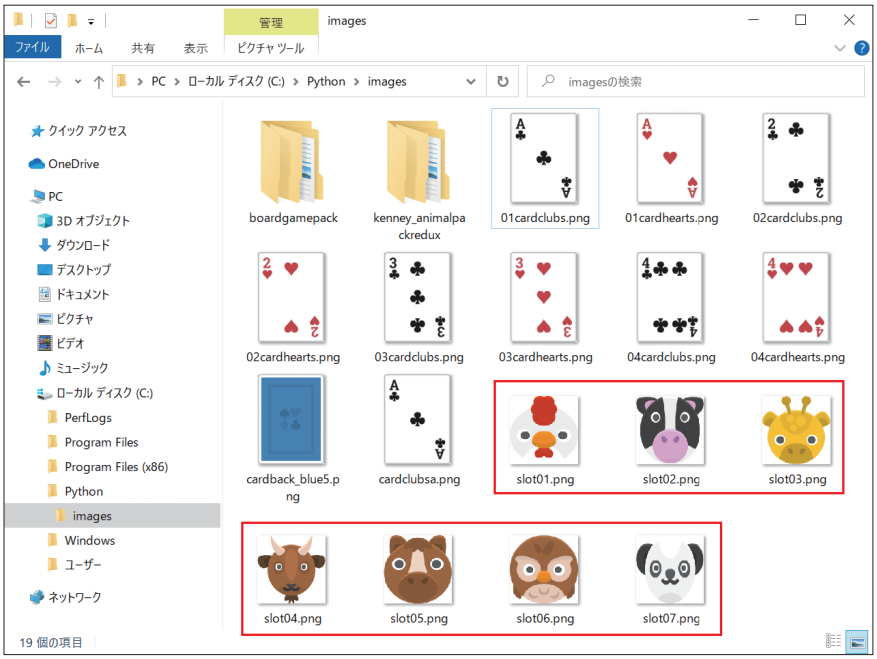
変換後のImageオブジェクトをsaveメソッドでファイルとして保存する

画像から生成したImageオブジェクトのresizeメソッドを呼び出し、100×100ピクセルに変更

**レイ** できたぁ。実行してみるわね…。

**タツヤ** おおお! slot01.pngからslot07.pngまで、全部100×100ピクセルのサイズになった（**図4**）。やっぱり、プログラムで変換するとはやいなぁ。

**図4●サイズを変更した画像ファイル**




**先生** 次は、スロットマシンのレバーとストップボタンの画像を用意しよう。

**レイ** いろんな画像が、ケニーのサイトにはあるわ。

**先生** 使えそうな画像がたくさんあるけど、今回は「Rolling Ball Assets」にあるものを使う。次のURLからダウンロードできるよ。

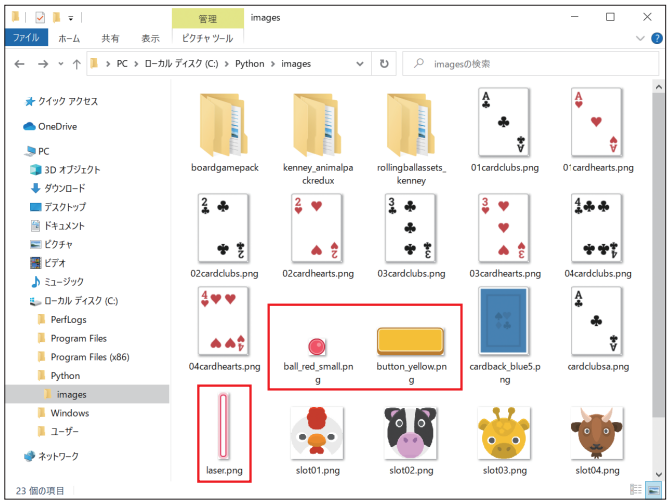
```
https://kenney.nl/assets/rolling-ball-assets
```

「rollingballassets_kenney.zip」をダウンロードしたら、imagesフォルダーに「rollingballassets_kenney」フォルダーを作る。その後、rollingballassets_kenneyフォルダーに、rollingballassets_kenney.zipの中にあるPNGフォルダーをコピーしよう。

**レイ** コピーできたよ。

**先生** そしたら、「PNG」→「Default」フォルダーにある「ball_red_small.png」と「button_yellow.png」、「laser.png」をimagesフォルダーの直下にコピーする（**図5**）。

**図5●レバーとボタンの画像ファイルを用意**




**タツヤ** ファイル名は小文字なので、そのままでOKだね?

**先生** その通り。ただし、button_yellow.pngのサイズが大きすぎるので、img_resize.pyを修正してサイズを100×40ピクセルに変更しておこう。

### img_resize.py

```
from PIL import Image

img = Image.open('./images/button_yellow.png')
img_resize = img.resize((100, 40))
img_resize.save('./images/button_yellow.png')
```

ファイル名を button_yellow.png に変更

サイズを100×40ピクセルに変更

**レイ** img_resize.py、大活躍ね。

**先生** それでは、画像をウインドウに配置する。これで、スロットマシンっぽくなるはずだ。次のコードを入力して、「animal_slot.py」で保存しよう。

### animal_slot.py

```
import pgzrun

WIDTH = 420
HEIGHT = 230

picture = []
picture.append(Actor('slot01', topleft=(30, 30)))
picture.append(Actor('slot02', topleft=(140, 30)))
picture.append(Actor('slot03', topleft=(250, 30)))
```

スロットの絵柄を表示するためのActorオブジェクト

## animal_slot.pyの続き

```
switch = Actor('laser', topleft=(375, 50))
ball = Actor('ball_red_small', topleft=(369, 50))

stop_button = []
stop_button.append(Actor('button_yellow', topleft=
(30, 150)))
stop_button.append(Actor('button_yellow', topleft=
(140, 150)))
stop_button.append(Actor('button_yellow', topleft=
(250, 150)))

def draw():
    for i in range(3):
        picture[i].draw()
        stop_button[i].draw()
    switch.draw()
    ball.draw()

pgzrun.go()
```

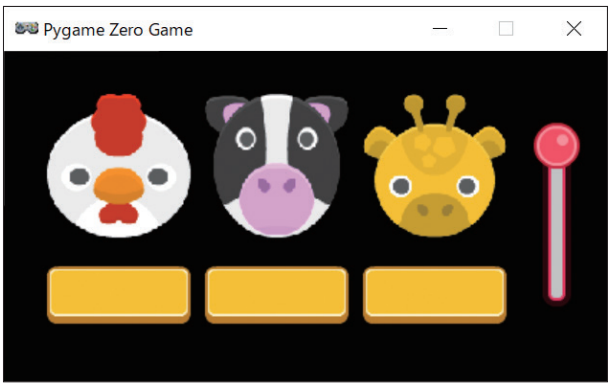
スロットレバー

ストップボタン

スロットマシンの表示

**タツヤ** よし、実行（図6）。

**図6●animal_slot.pyの実行結果**




**レイ** マジかわいいんですけど。

**タツヤ** う〜ん、スロットマシンと言えば、言えなくもないかな?

**先生** では、この画面でスロットマシンの中身を作っていくことにする。

# 3日目 Part 2 スロットの回転

**先生** 画面ができたので、ゲームの処理を考えよう。まずは、レバーのボールをクリックしたら「スロットの画像」、つまり、スロットマシンのドラムに描かれた画像を自動的に切り替えたい。どうすればいいだろうか?

**レイ** レバーのボールをクリックしたかどうかは、マウスクリックで呼び出される関数を使えばいいんじゃない? イベント“パンドラ”だっけ? その中に画像を入れ替える処理を書けばいいと思う。

**先生** イベント“ハンドラ”ね。ただ、それだとクリックしたときしか画像が切り替わらないよ。

**タツヤ** そうか、update関数で切り替えるんだ。

**先生** その通り。ただ、ドラムの画像は3つある。それぞれストップボタンで止まるようにしなければならない。

**レイ** if文でドラムが回転しているかどうかを調べればいいんじゃない? もし、回転中なら画像を切り替えるし、ストップボタンが押されてたら切り替えない…。今の状態を覚えておく変数が必要ね。

**先生** そこで、3つのグローバル変数を用意しようと思う。

animal_slot.py

```
import pgzrun

WIDTH = 420
HEIGHT = 230

picture = []
picture.append(Actor('slot01', topleft=(30, 30)))
picture.append(Actor('slot02', topleft=(140, 30)))
picture.append(Actor('slot03', topleft=(250, 30)))
```

## animal_slot.pyの続き

```
switch = Actor('laser', topleft=(375, 50))
ball = Actor('ball_red_small', topleft=(369, 50))

stop_button = []
stop_button.append(Actor('button_yellow', topleft=(30,
150)))
stop_button.append(Actor('button_yellow', topleft=(140,
150)))
stop_button.append(Actor('button_yellow', topleft=(250,
150)))

slot_count = [1, 2, 3] # 見えている絵柄
drum = [1, 1, 1]       # 0:ドラム回転中、1:ドラムストップ中
time_count = 0         # 経過時間のカウント用（60で1秒）

def draw():
    screen.clear()
    switch.draw()
    ball.draw()
    for i in range(3):
        picture[i].draw()
        stop_button[i].draw()
```

追加するグローバル変数

**レイ** slot_countやdrumはリストですね。

**先生** slot_countは、見えている絵柄の番号。drumは、ドラムが回転中かどうかを保持するためのもので、0なら回転中、1なら止まっている。どちらも、3つ必要なのでリストにしている。

**タツヤ** time_countは?

**先生** update関数は、1秒間に60回実行される。もし、update関数が実行されるたびに画像を切り替えてしまうと、速すぎてスロットマシンに見えない。そこで、切り替える時間を調整するためにtime_countを使う。

**タツヤ** つまり、レバーのボールをクリックしたらドラムを全部「ドラム回転中」に切り替える。で、ストップボタンをクリックしたら、対応する変数を「ドラムストップ中」に変更する、ってことか…。

**レイ** update関数では、ドラムの今の状態を調べて、画像を切り替えるのね。

**先生** そういうこと。それじゃ、演習時間だ。紹介したコードを参考にして、各自、スロットマシンの基本部分をコーディングしてみよう。

### animal_slot.py（その2）

```
def on_mouse_down(pos):
    global slot_count
    if ball.collidepoint(pos):
        for i in range(3):
            drum[i] = 0
    else:
        for n in range(3):
            if stop_button[n].collidepoint(pos):
                drum[n] = 1

def update():
    global slot_count
    global time_count
    time_count +=1
    if time_count == 20:
        time_count = 0
        for i in range(3):
            if drum[i] == 0:
                slot_count[i] += 1
                if slot_count[i] == 8:
                    slot_count[i] = 1
                picture[i].image = 'slot0' + str(slot_coun
t[i])

pgzrun.go()
```

ボールをクリックした

すべてのドラムを回転中にする

対応するドラムを止める

ストップボタンをクリックした

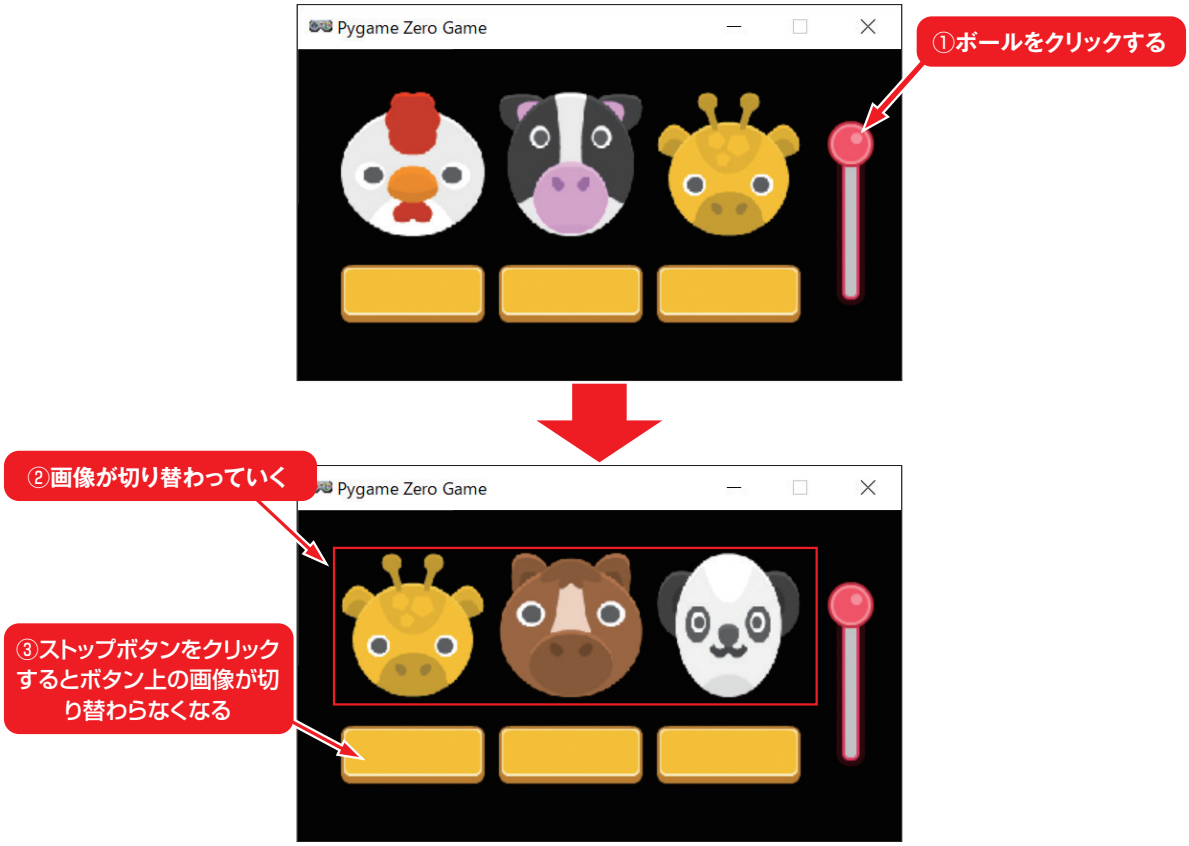
約0.33秒に1回切り替える

ドラムが回転中なら次の画像へ切り替える

. . .

**レイ** これでいいかな? 実行してみよ（図1）。

**図1●animal_slot.pyの実行結果**




**タツヤ** おおお! レバーのボールをクリックしたら自動的に画像が切り替わる! んで、ストップボタンで止まる…。けど、なんかダサくね?

**レイ** やっぱり、パラパラ切り替わるんじゃなくて、回転しているように見せたいわね。

**先生** そうか…。なら、ドラムの絵柄を下方向へ徐々に移動させるのはどうだろう。

**レイ** 画像が下へ移動するだけか〜。もうひと捻り欲しいような…。

**先生** そういうときは、実際のスロットマシンがどのような構造になっているのかを考えてみよう。スロットマシンは、ドラム側面の一部分だけが筐体の「窓」から見えていて、それ以外の部分は見えていないはずだ。

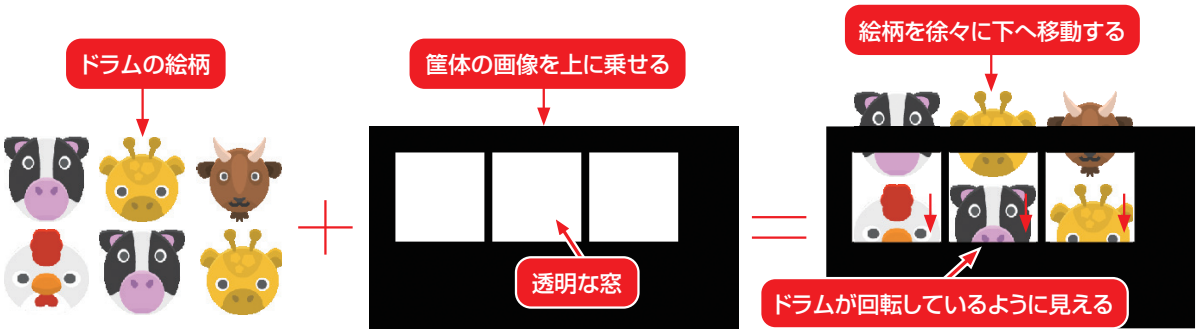
**タツヤ** じゃあ、ドラムの上に、筐体の画像を貼り付けるってのはどう?

**レイ** なにそれ?

**先生** タツヤ君、ナ・イ・ス。つまり、窓の部分だけを透明にした筐体の画像を用意して、その画像をドラムの絵柄の上に重ねる。その上で、絵柄を下に移動させる。こうすれば、よりスロットマシンっぽくなると思う。

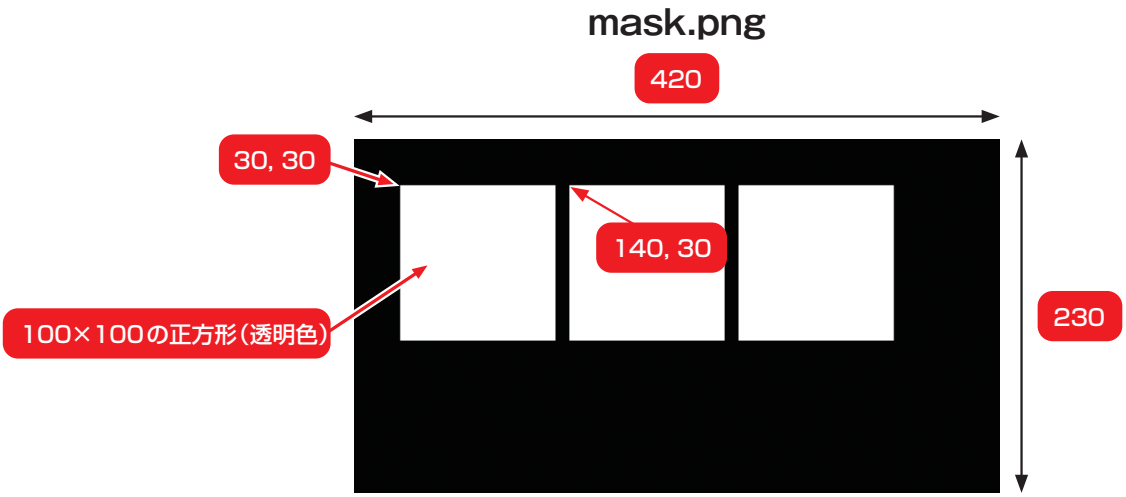
**タツヤ** 絵柄は、1枚目の画像の上に2枚目の画像を表示して、一緒に下へ動かせば、あたかもドラムが回転しているように見えるのじゃあ(**図2**)。

**図2●ドラムが回転しているように見せる**




**レイ** なんかぁ、盛り上がってますけどぉ…。その「窓が開いた筐体の画像」(**図3**)って、どうやって用意するんですかぁ?

**図3●窓が開いた筐体の画像**




**タツヤ** これなら、お絵描きソフトで作れそうだ。

**先生** 確かにお絵描きソフトを使っても作れるけど、せっかくなのでプログラムで生成しよう。次のコードをIDLEのエディターで入力し、「make_mask.py」の名前でPythonフォルダーに保存して、実行する。

### make_mask.py


```
from PIL import Image, ImageDraw

img_mask = Image.new('RGBA', (420, 230), (0, 0, 0, 255))
draw = ImageDraw.Draw(img_mask)
draw.rectangle((30, 30, 130, 130), fill=(255, 255, 255, 0))
draw.rectangle((140, 30, 240, 130), fill=(255, 255, 255,
0))
draw.rectangle((250, 30, 350, 130), fill=(255, 255, 255,
0))
img_mask.save('./images/mask.png')
```

**レイ** 「マスクを作ろう」ってプログラムなの? いまどき〜。

**先生** このように、画像の特定の部分のみを表示して、それ以外の部分を表示しないようにする画像処理のことを「マスク処理」という。プログラム名は、ここからきている。

**レイ** make_mask.pyを実行したら、imagesフォルダーに「mask.png」ができたわ。

**タツヤ** さぁ、後は絵柄を回転させるプログラムを作るだけだ。

# 3日目 Part 3 ドラムの回転とストップ

**先生** それでは、ドラムの回転とストップのコードを考えよう。

**タツヤ** 思ったんだけど、最初から7枚の画像を縦に並べた1枚の画像にしておく方が楽じゃないかなぁ?

**先生** もちろん。ただ、そうするといつも絵柄の順番が同じなので、何度もプレイしていると「目押し」がしやすくなってしまう。

**レイ** 目押し?

**先生** 動いている絵柄を見つめて、次に来る絵柄を狙って止めるテクニックのことだ。だから、やはりバラバラの画像をプログラムでつなげて表示した方がゲームの難易度を調整しやすい。

**タツヤ** 仕方ない。ドラムを表現するリストとか作るしかないかぁ…。

**先生** 先ほど、ドラムが回転しているか否かの判断用にdrumという名前を使ってしまった。なので、絵柄を保持するドラム3本は「wheel」という2次元のリストに格納することにしよう。

**レイ** wheelに3本のドラムを格納して、各ドラムに7枚の画像を貼り付けるイメージね。

**先生** そうだ。次に、slot_countも変更する。1枚目の画像とその上にくる2枚目の画像を扱うので、こちらも2次元のリストにした。

**タツヤ** 7枚のうち、プレイヤーに見えるのは2枚だからね。

**先生** それでは、ゲームに登場するオブジェクトの宣言と初期化のコードから紹介しよう。

## animal_slot.py（宣言と初期化部分）

```
import pgzrun

WIDTH = 420
HEIGHT = 230

wheel = [[],[],[]]
for i in range(1, 8):
    wheel[0].append(Actor('slot0' + str(i), center=(80,
0)))
    wheel[1].append(Actor('slot0' + str(i), center=(190,
0)))
    wheel[2].append(Actor('slot0' + str(i), center=(300,
0)))

for n in range(3):
    wheel[n][n].y = 80
    wheel[n][n + 1].y = -20

mask = Actor('mask', topleft=(0, 0))
switch = Actor('laser', topleft=(375, 50))
ball = Actor('ball_red_small', topleft=(369, 50))

stop_button = []
stop_button.append(Actor('button_yellow', topleft=(30,
150)))
stop_button.append(Actor('button_yellow', topleft=(140,
150)))
stop_button.append(Actor('button_yellow', topleft=(250,
150)))

slot_count = [[0,1], [1,2], [2,3]] # 見えている絵柄
drum = [1, 1, 1]     # 0:ドラム回転中、1:ドラムストップ中
```

y座標は後から指定する

3つのドラムを束ねたwheelリスト

y座標を指定して配置

マスク画像とレバー

窓から見える部分は各ドラムごとに2枚の絵柄が必要

**レイ** リストだらけね。

**タツヤ** でも、おかげでプログラムはシンプルになっている。

**先生** スロットマシンに必要な部品をすべて用意できたら、draw関数で表示する。マスク画像の表示は、wheelを表示した後になるので注意すること。

### animal_slot.py（draw関数）

```
def draw():
    screen.clear()
    for i in range(3):
        wheel[i][slot_count[i][0]].draw()
        wheel[i][slot_count[i][1]].draw()
    mask.draw()
    switch.draw()
    ball.draw()
    for n in range(3):
        stop_button[n].draw()
```

ドラムを表示

マスク画像を重ねる

**レイ** オブジェクトを表示する順番が大切ね。

**先生** マウスクリックの処理は、基本、変更はない。レバーのボールがクリックされたら、drumを0にして「回転中」にする。ストップボタンがクリックされたら1にして「ストップ」にするだけだ。

### animal_slot.py（on_mouse_down関数）

```
def on_mouse_down(pos):
    if ball.collidepoint(pos):
        for i in range(3):
            drum[i] = 0 # ドラムを回転中にする
    else:
        for n in range(3):
            if stop_button[n].collidepoint(pos):
                drum[n] = 1 # 回転を止める
```

**レイ** ここら辺はもう簡単ね。

**先生** 要になるのがupdate関数だ。ここでは、回転させる必要があるドラムなら、オリジナルのrotation関数を呼び出すようにした。

animal_slot.py（残り部分）

```
def update():
    for i in range(3):
        if drum[i] == 0: # ドラムが回転中なら
            rotation(i)


def rotation(num):
    global slot_count
    wheel[num][slot_count[num][0]].y += 1
    wheel[num][slot_count[num][1]].y += 1
    if wheel[num][slot_count[num][0]].y > 180: # 境目
        slot_count[num][0] += 1
        slot_count[num][1] += 1
        if slot_count[num][0] == 6:
            slot_count[num][1] = 0
        elif slot_count[num][0] == 7:
            slot_count[num][0] = 0
        wheel[num][slot_count[num][0]].y = 80
        wheel[num][slot_count[num][1]].y = -20

pgzrun.go()
```

回転しているドラムだけを対象にする

ドラムの絵柄を下に移動

表示する絵柄の番号を入れ替える

回転しているドラムだけを対象にする

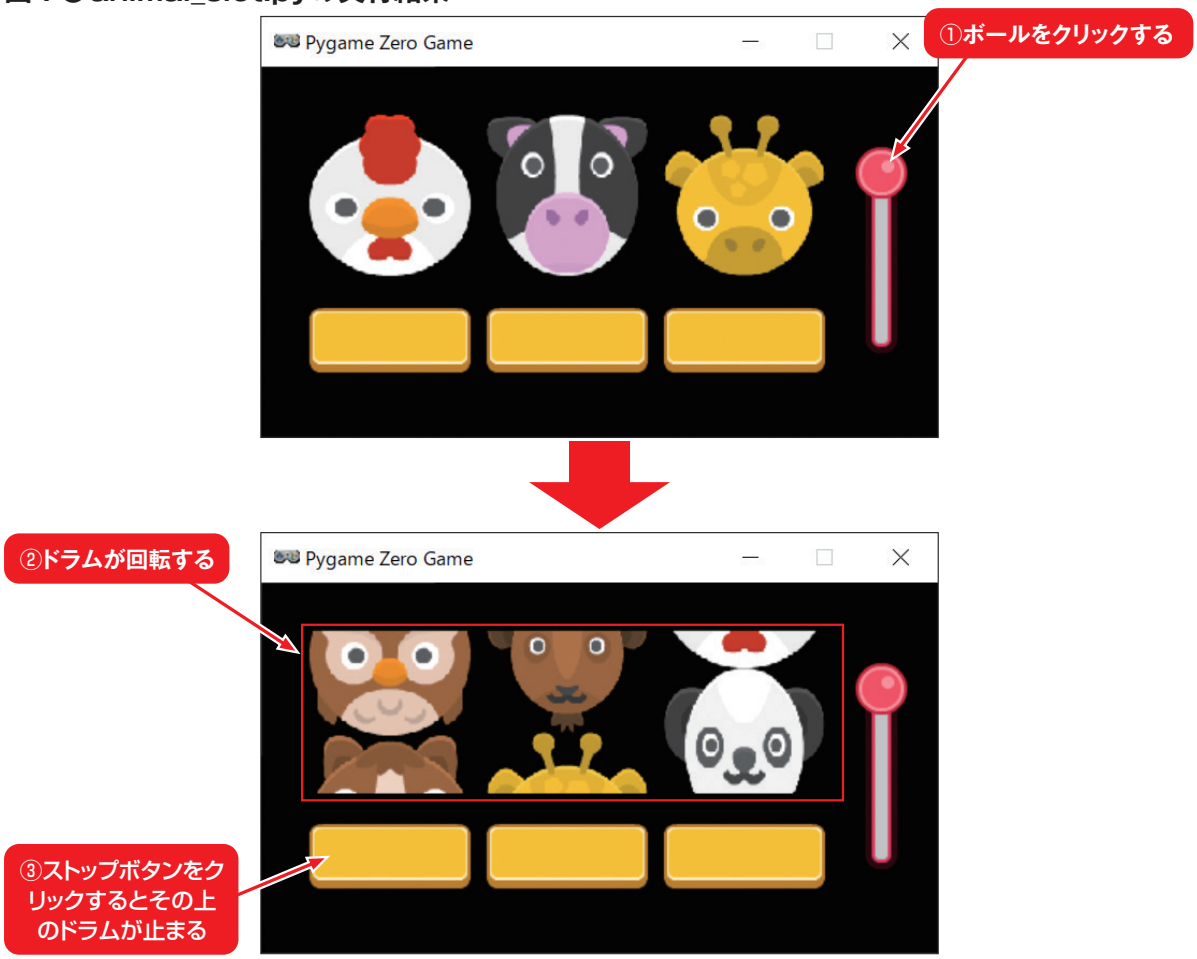
1枚目の絵柄が下に移動して見えなくなったタイミング

**レイ** 今度は、if文の嵐ね…。

**タツヤ** う〜ん、そうか。基本はドラムの絵柄をy座標で下に移動させるんだけど、もし1枚目の絵柄が見えなくなったら、その上の2枚目の画像を1枚目と入れ変えて同じように下げていくんだな。

**先生** あとは、絵柄が7枚目の時に1枚目に戻すことで、ループしているように見せている。入力できたら、実行してみよう（**図1**）。

図1●animal_slot.pyの実行結果




**タツヤ** なるほど…。確かにドラムが回転しているように見えるから不思議だぁ。でも、ストップボタンをクリックするといきなり止まっちゃうんだなぁ。このままじゃ、まだスロットマシンって感じがしないなぁ。

**レイ** それから、レバーもクリックしたら下がるアニメーションがあるといいわね。ボタンもへこまないから、違和感あるわぁ。

**先生** もう少しで完成だが、今日はここまでにしよう。

**タツヤ** イヤ、俺はこの後も作り続けるぜ…。

**先生** それは構わないけどあまり無理しないように…。では、また明日。

4日目

# スロットゲームの完成と荷物運びパズルゲームの準備

# 4日目 Part 1 アニメーションのプログラミング

**先生** さぁ、今日はスロットゲームの完成と、明日の「荷物運びパズルゲーム」の画面部分を作ることが目標だ。まずは、スロットゲームからだ。

**レイ** ドラムの回転とストップはできたけど、クリックすると絵柄の途中でも止まってしまう。これじゃ、だめよねぇ。

**タツヤ** ボタンもレバーもリアルじゃないし…。

**先生** まぁまぁ。まだ、未完成だからしょうがない。最初はタツヤ君の要望である「ボタンとレバーのリアル化」を考えよう。

**タツヤ** やったぜぇ。

**先生** ボタンとレバーがリアルに感じないのは、クリックしたときに動きがないからだ。例えば、ボタンはへこまないといけないし、レバーは下がらないと変だ。

**レイ** でも、アニメーションをプログラミングするのは大変じゃない?

**先生** そうだね。なので、できるだけ少ない労力で最大限の効果を狙おう。

**タツヤ** と、言いますと?

**先生** ボタンに関しては、へこんだ時の画像を別途用意して、クリックされたときに入れ替えることで動きを与える。レバーは、Pygame Zeroが用意している「animate関数」で、レバーのボールを、変化のある動きで下まで移動させよう。

**レイ** animate関数は、前回作った「月面着陸ゲーム」でちょっと使ったやつね。へこんだボタンの画像もプログラムで作るの?

**先生** こちらも、お絵描きソフトを使って画像を用意してもいいのだけど、次の「make_button.py」というプログラムを作ったので入力して実行してみよう。

make_button.py

文字を画像に描くために必要なImageFont

button_yellow.pngを読み込み、透明色を含むImageへ変換

```
from PIL import Image, ImageFont, ImageDraw

img = Image.open('./images/button_yellow.png').convert('RGBA')
font = ImageFont.truetype('C:¥¥Windows¥¥Fonts¥¥bahnschrift.ttf', 24)
draw = ImageDraw.Draw(img)
draw.text((20, 7), 'STOP', font=font, fill=(0,0,0,255))
img.save('./images/stop_button1.png')

img_push = Image.new('RGBA', (100, 40), (0, 0, 0, 255))
img_crop = img.crop((0, 0, 98, 36))
img_push.paste(img_crop, (2, 5))
img_push.save('./images/stop_button2.png')
```

描画する文字のフォントとサイズを指定

ボタンのImageに「STOP」と黒で描く

stop_button1.pngで保存

ボタンの上層部を切り取る

100×40の長方形を黒で描く

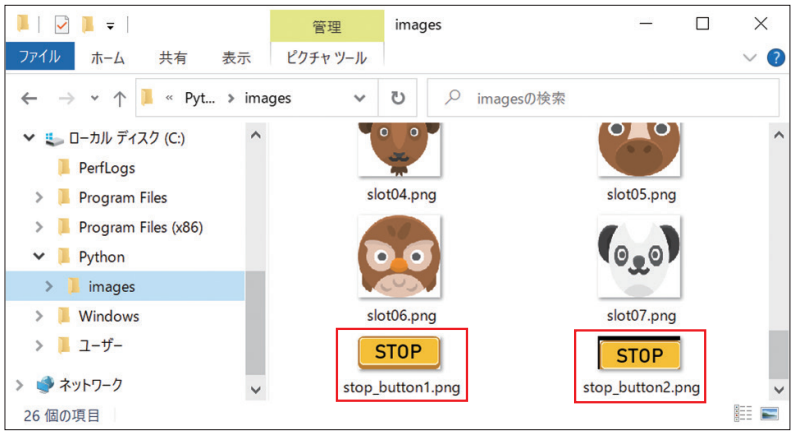
stop_button2.pngで保存

長方形の右斜め下に貼り付けてへこんだボタンを作成

**レイ** よし、実行!

**タツヤ** 「stop_button1.png」と、へこんだ「stop_button2.png」ができた。STOPって文字も入ってる(**図1**)。

**図1●make_button.pyで生成したボタン**



**先生** プログラムを少し解説しよう。最初にbutton_yellow.pngを読み込んで、「STOP」という文字を描いたら、stop_button1.pngで保存する。次に、ボタンと同じサイズの長方形を黒で描いておき、stop_button1.pngのボタン上層部を切り取って、長方形の斜め右下に貼り付ける。このイメージをstop_button2.pngで保存する。

**レイ** お絵描きソフトで行う作業を、プログラムにやらせてる感じね。

**先生** それではこのストップボタンの画像を使って、クリックされたらへこむようにプログラムを変更できるかな?

. . .

**レイ** できたわ。簡単ね。

### animal_slot.pyの修正

```
. . .

stop_button = []
stop_button.append(Actor('stop_button1', topleft=(30,
150)))
stop_button.append(Actor('stop_button1', topleft=(140,
150)))
stop_button.append(Actor('stop_button1', topleft=(250,
150)))

. . .

def on_mouse_down(pos):
    if ball.collidepoint(pos):
        for i in range(3):
            drum[i] = 0 # ドラムを回転中にする
    else:
        for n in range(3):
            if stop_button[n].collidepoint(pos):
                if drum[n] == 0:
                    stop_button[n].image = 'stop_button2'
                    drum[n] = 1
```

ストップボタンのファイル名を変更

ストップボタンがクリックされた？

ドラムが止まっていたらstop_button2.pngに変更

**先生** 今度は、animate関数を使って、レバーのボールをクリックしたら、ボールがレバーの下まで移動するように変更してみよう。

. . .

**タツヤ** これも簡単だった。アニメーションのやり方は、Pygame Zeroのマニュアルを見ればわかったよ。どんな動きにするのかは、引数のtweenに与える文字列で決まるみたい。いろいろ試した結果「'bounce_end'」にしたよ。

**animal_slot.pyのon_mouse_down関数**

ボールをレバーの下まで1秒間のアニメーションで移動

```
def on_mouse_down(pos):
    if ball.collidepoint(pos):
        animate(ball, pos=(385, 165),tween='bounce_end', d
uration=1)
        for i in range(3):
            drum[i] = 0 # ドラムを回転中にする
    else:
        for n in range(3):
            if stop_button[n].collidepoint(pos):
                if drum[n] == 0:
                    stop_button[n].image = 'stop_button2'
                    drum[n] = 1
```

**レイ** 変更したプログラムを実行してみよう（**図2**）。

**図2●変更したプログラムの動作**


**先生** これで、それなりにリアルな動きのスロットマシンになってきた。では、いよいよ絵柄の境目で止まるようにする処理に取りかかろう。

**タツヤ** ストップボタンをクリックしたタイミングと、実際に止まるタイミングが違うわけだけど…。どうやるの?

**レイ** ボタンがクリックされたときに、絵柄の「境目」なら止める。そうでなければ回し続けて「境目」に来たら止めるのよ。

**タツヤ** だから、どうやって?

**レイ** そんなこと、知らないわよ!

**先生** レイちゃんの方法を採用するとして、ドラムの「ストップ要求」を調べるグローバル変数が必要だね。

**タツヤ** ストップ要求…。いきなりドラムを止めるんじゃなくて、「境目で止めてください」って「要求する」ってこと?

**先生** そう。サンプルコードを用意したので、まずは入力して動きを確認してみよう。

### animal_slot.pyの修正

```
...

stop_flg = [0, 0, 0]   # 0:回転中、1:ストップ要求中

...

def on_mouse_down(pos):
    if ball.collidepoint(pos):
        animate(ball, pos=(385, 165),tween='bounce_end',
duration=1)
        for i in range(3):
            drum[i] = 0 # ドラムを回転中にする
    else:
        for n in range(3):
            if stop_button[n].collidepoint(pos) and drum[n]
 == 0:
                stop_button[n].image = 'stop_button2'
                stop_flg[n] = 1 # ドラムのストップを要求する

...

def rotation(num):
    global slot_count
    wheel[num][slot_count[num][0]].y += 4
    wheel[num][slot_count[num][1]].y += 4
    if wheel[num][slot_count[num][0]].y > 180 and stop_flg[
num] == 1: # 境目かつストップ要求あり
        drum[num] = 1 # 本当にドラムを止める
    elif wheel[num][slot_count[num][0]].y > 180:
        slot_count[num][0] += 1
        slot_count[num][1] += 1
        if slot_count[num][0] == 6:
            slot_count[num][1] = 0
        elif slot_count[num][0] == 7:
            slot_count[num][0] = 0
        wheel[num][slot_count[num][0]].y = 80
        wheel[num][slot_count[num][1]].y = -20
```

「ストップ要求」が発生しているのかどうかを調べるためのグローバル変数

回転スピードはここで調整

境目かつ「ストップ要求」が発生しているときはドラムを止める

**タツヤ** できた（**図3**）。要は「ストップボタンがクリックされたときの処理」と「実際に境目で止める処理」を分ける、って部分がポイントだね。

**図3●境界部分まで回転して止まるようになった**


**レイ** その橋渡しを、グローバル変数のstop_flgが行うのね。

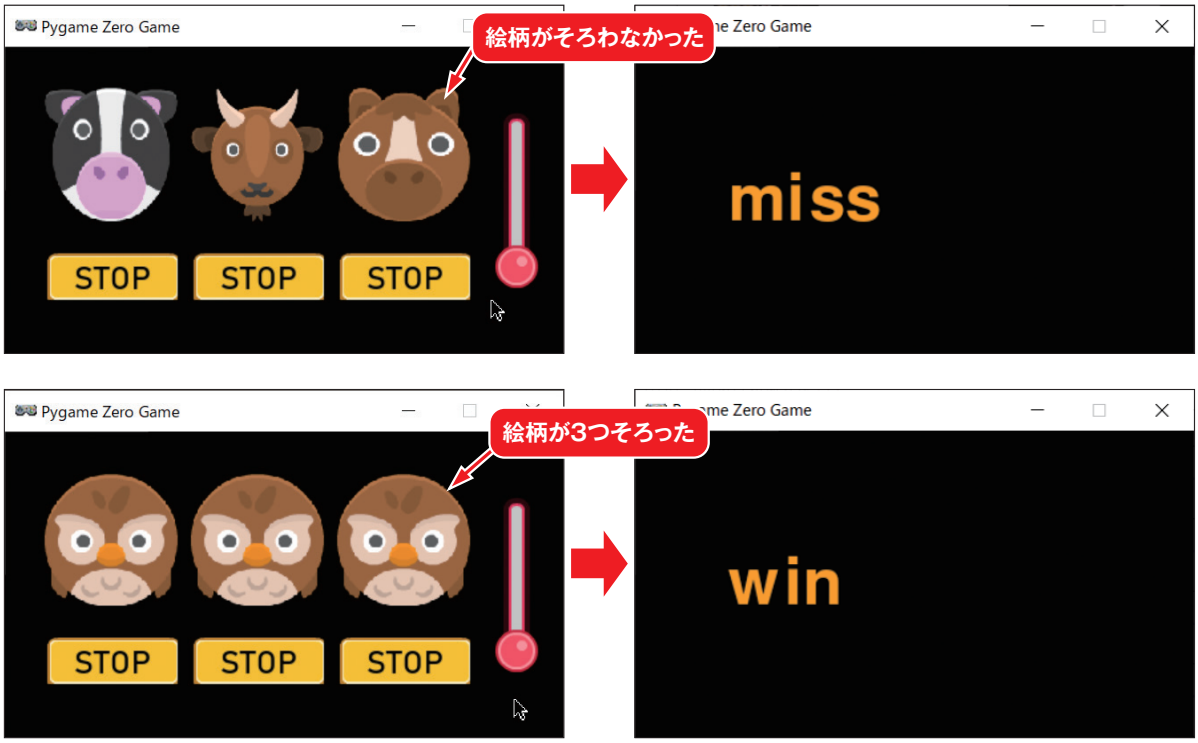
## 4日目 Part 2 スロットゲームの完成

**先生** さぁ、いよいよスロットゲームを完成させよう。

**タツヤ** 今の段階で結構ゲームらしくなってきたよね。

**先生** スロットマシンは、スロットの絵柄が3つそろうと「当たり」、そろわなければ「はずれ」だ。そこで、3つの絵柄が決まったら、「当たり」か「はずれ」を表示することにしよう。これで、ゲームとしての最低限の機能はそろう（**図1**）。

**図1●当たり、はずれの判定を追加して完成**




**レイ** 月面着陸ゲームでやったみたいに、現在、ゲームがどういう状態にあるのかを覚えておく必要があるわね。

**タツヤ** statusって変数だったかな。1ならスタート画面、2ならゲーム中、3ならゲームオーバー、4ならゲームクリアみたいに数値でゲームの状態を表現するんだね。

**先生** それでは、サンプルのコードを見せるので、各自でスロットゲームを完成させて欲しい。

**animal_slot.py**

```
import pgzrun

WIDTH = 420
HEIGHT = 230

wheel = [[],[],[]]
for i in range(1, 8):
    wheel[0].append(Actor('slot0' + str(i), center=(80,
```

## animal_slot.pyの続き

```
0)))
    wheel[1].append(Actor('slot0' + str(i), center=(190,
0)))
    wheel[2].append(Actor('slot0' + str(i), center=(300,
0)))

for n in range(3):
    wheel[n][n].y = 80
    wheel[n][n + 1].y = -20

mask = Actor('mask', topleft=(0, 0))
switch = Actor('laser', topleft=(375, 50))
ball = Actor('ball_red_small', topleft=(369, 50))

stop_button = []
stop_button.append(Actor('stop_button1', topleft=(30,
150)))
stop_button.append(Actor('stop_button1', topleft=(140,
150)))
stop_button.append(Actor('stop_button1', topleft=(250,
150)))

slot_count = [[0,1], [1,2], [2,3]] # 見えている絵柄
drum = [1, 1, 1]                   # 0:ドラム回転中、1:ドラムストップ中
stop_flg = [0, 0, 0]               # 0:回転中、1:ストップ要求中
status = 1             # 1:開始画面、2:ゲーム中、3:終了、4:判定表示
message = ''                       # 判定結果メッセージ
time_count = 0                     # 判定表示までの時間稼ぎ用の変数

def draw():
    screen.clear()
    if status == 4:
        screen.draw.text(message, (70, 90), color='oran
ge', fontsize=72)
    else:
        for i in range(3):
            wheel[i][slot_count[i][0]].draw()
            wheel[i][slot_count[i][1]].draw()
        mask.draw()
        switch.draw()
        ball.draw()
        for n in range(3):
```

## animal_slot.pyの続き

```
            stop_button[n].draw()

def on_mouse_down(pos):
    global status
    if status == 1:
        if ball.collidepoint(pos):
            animate(ball, pos=(385, 165),tween='bounce_
end', duration=1)
            for i in range(3):
                drum[i] = 0
            status = 2 # ゲーム中へ
    elif status == 2:
        for n in range(3):
            if stop_button[n].collidepoint(pos) and drum
[n] == 0:
                stop_button[n].image = 'stop_button2'
                stop_flg[n] = 1 # ドラムのストップを要求する

def update():
    global status
    global time_count
    if status == 3: # ゲーム終了なら
        time_count += 1
        if time_count == 120: # 2秒待ってから
            judgment() # 判定表示の準備関数へ
    elif status == 2:
        for i in range(3):
            if drum[i] == 0:
                rotation(i)

def rotation(num):
    global slot_count
    global status
    wheel[num][slot_count[num][0]].y += 4
    wheel[num][slot_count[num][1]].y += 4
    # 境目でかつストップ要求あり
    if wheel[num][slot_count[num][0]].y > 180 and stop_flg[
num] == 1:
        drum[num] = 1
        if drum[0] == 1 and drum[1] == 1 and drum[2] == 1:
            # ボールを戻す
            animate(ball, pos=(385, 50),tween='bounce_
```

## animal_slot.pyの続き

```
end', duration=1)
            for i in range(3):
                stop_button[i].image = 'stop_button1' # ボ
タンを戻す
            status = 3 # ゲーム終了へ
    elif wheel[num][slot_count[num][0]].y > 180:
        slot_count[num][0] += 1
        slot_count[num][1] += 1
        if slot_count[num][0] == 6:
            slot_count[num][1] = 0
        elif slot_count[num][0] == 7:
            slot_count[num][0] = 0
        wheel[num][slot_count[num][0]].y = 80
        wheel[num][slot_count[num][1]].y = -20

def judgment(): # 判定表示の準備関数
    global status
    global message
    # 判定
    j = slot_count[0][0] == slot_count[1][0] == slot_cou
nt[2][0]
    # 判定結果のメッセージを作成
    message = 'win' if j else 'miss'
    status = 4 # 判定表示へ

pgzrun.go()
```

# 4日目 Part 3 荷物運びパズルゲームを作る

**先生** いよいよ、続・Pythonゲームコースで作る最後のゲーム「荷物運びパズルゲーム」にとりかかる。荷物運びパズルゲームは、「倉庫」の中にいる「プレイヤー」を操作して、「障害物の箱」をかわしながら「荷物」を「出口」まで運ぶゲームだ。

**タツヤ** 倉庫の中でプレイヤーを動かすと言えば、前回、迷路を作ったけど、似た感じ?

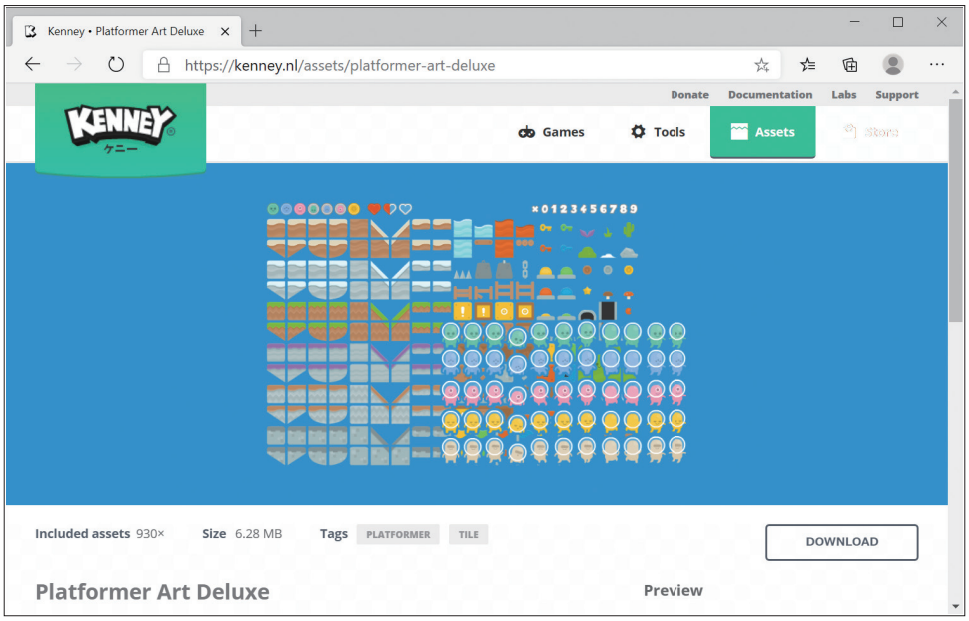
**先生** そう。キーボードを使ってプレイヤーを上下左右に動かす仕組みは同じだ。荷物運びパズルゲームでは、迷路プログラムの画面を流用しようと思う。そして、プレイヤーが荷物を押しながら移動する処理を追加する。

**レイ** 迷路…。じゃ、エイリアンさんもダウンロードしなきゃ!

**先生** それでは、ケニーのサイトから「Platformer Art Deluxe」(**図1**)をダウンロードするところから始めよう。次のURLでPlatformer Art Deluxeのページにアクセスできる。

```
https://kenney.nl/assets/platformer-art-deluxe
```

**図1●Platformer Art Deluxeをダウンロード**



**レイ** 「platformer-art-complete-pack-0.zip」をダウンロードできたよ。

**先生** そしたら、imagesフォルダーに「platformer-art-completepack-0」フォルダーを作って、platformer-art-complete-pack-0.zipの中身をすべて、platformer-art-completepack-0フォルダーにコピーしよう。

**タツヤ** 丸ごとコピーっと。

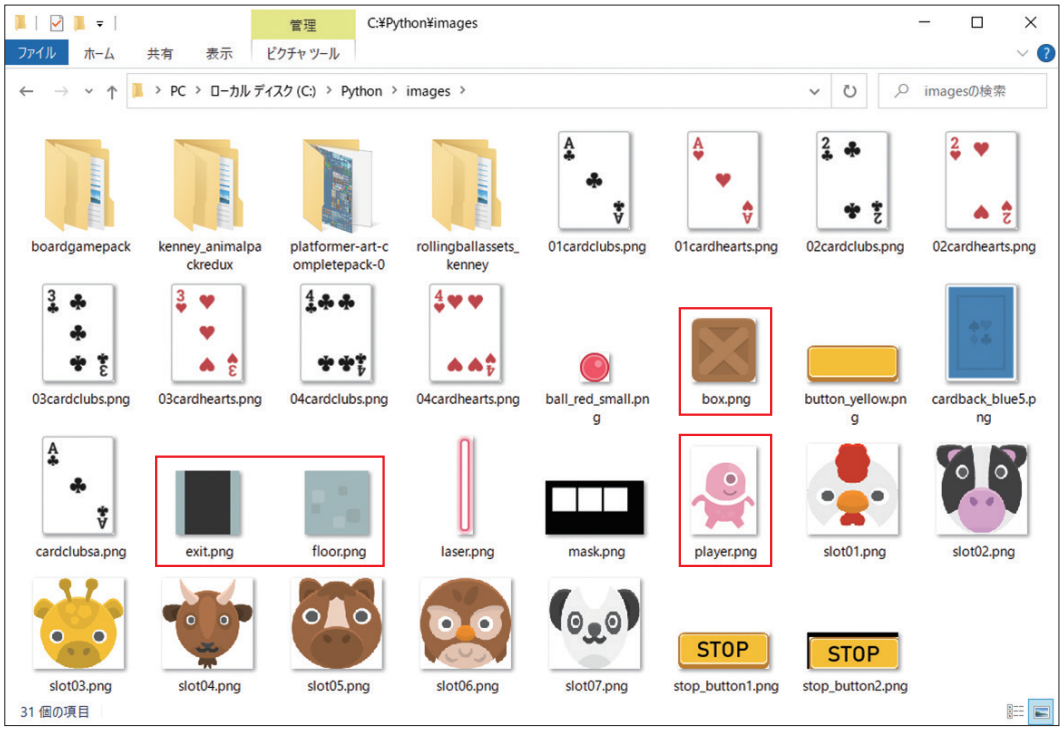
**先生** コピーできたら、「Base pack」→「Player」→「p3_walk」→「PNG」とフォルダーをたどってみよう。エイリアンの画像があるよね。この中にある「p3_walk03.png」をimagesフォルダーの直下にコピーする。

**レイ** エイリアンさん、なつかしいなぁ。

**先生** 次は背景や障害物などの画像だ。platformer-art-complete-pack-0フォルダーにある「Base pack」→「Tiles」フォルダーの中に、「boxAlt.png」と「castleCenter.png」、「door_openMid.png」がある。これらをimagesフォルダーにコピーする。そして、コピーした計4個のファイル名を次のように変更しよう（**図2**）。

```
p3_walk03.png     → player.png   （プレイヤー）
boxAlt.png        → box.png      （障害物の箱）
castleCenter.png  → floor.png    （倉庫の床）
door_openMid.png  → exit.png     （出口）
```

**図2●ゲームに使用する画像ファイルを用意**




**タツヤ** 迷路は確か、2次元のリストだっけ?

**先生** その通り。2次元のリストを作って、障害物の「ある」「なし」を数値の1か0で表現する。その場合、リストのインデックスの値を障害物のx座標とy座標にできる。例えば、前回作った迷路のプログラムでは、次のような2次元リストで迷路を表現していた(**図3**)。

**図3●2次元リストで迷路を表現**

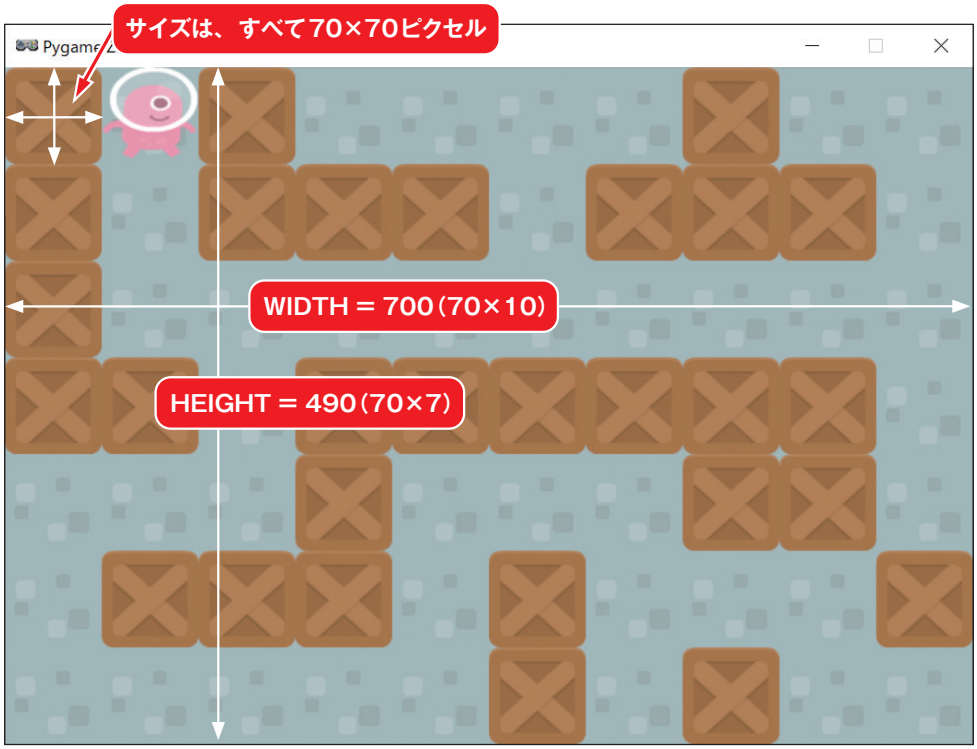
```
map_data =[[1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 0, 1, 1, 1, 1, 1, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
           [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]
```

**レイ** これ、ビジュアル的にもわかりやすいよね〜。

**先生** この2次元リストが、**図4**の迷路になった。なお、迷路の周りには「壁」があること

にするよ。

**図4●迷路の画面。迷路の外周には壁があることにする**




**レイ** そうだった、そうだった。1つのキャラクターの画像を70×70ピクセルに統一することで、配置や移動の処理が簡単になるのよね。

**先生** そうだね。そこでimg_resize.pyを修正して、player.pngを70×70ピクセルのサイズに変更しておこう。ほかの3つの画像ファイルは、最初から70×70ピクセルなので、そのままで大丈夫だ。

**img_resize.py**

```
from PIL import Image

img = Image.open('./images/player.png')
img_resize = img.resize((70, 70))
img_resize.save('./images/player.png')
```

ファイル名を player.png に変更

サイズを70×70ピクセルに変更

**先生** 最後に、倉庫を描画するサンプルコードを紹介しよう。といっても、迷路のプログラムとほとんど同じなので、思い出しながら入力して「carry_cargo.py」のファイル名で保

存しよう。

carry_cargo.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

map_data =[[1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 1, 0, 1, 1, 1, 1, 1, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
           [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]

player = Actor('player', topleft=(70, 0))
floor = Actor('floor', topleft=(0, 0))
box = Actor('box', topleft=(0, 0))
exit = Actor('exit', topleft=(630, 420))

def draw():
    screen.clear()
    for y in range(7):
        for x in range(10):
            floor.topleft=(70*x, 70*y)
            floor.draw()
            if map_data[y][x] != 0:
                box.topleft=(70*x, 70*y)
                box.draw()
    exit.draw()
    player.draw()

pgzrun.go()
```

タツヤ 実行! コレだコレ（図5）。さぁ、早くエイリアンを動かそう。

図5●carry_cargo.pyの実行例


先生 今日は、ここまでにしておく。

タツヤ なに!

レイ タツヤは、この後も続ける気でしょう。

先生 明日は最終日だから、あまり無理はしないように。それではオンラインを切るよ〜。

タツヤ レイ さようなら〜。

5日目

# 荷物運びパズルゲームを完成させよう

# 5日目 Part 1 プレイヤーを動かす

**先生** さて、昨日は「荷物運びパズルゲーム」の倉庫の画面を表示するところまで説明したが、大丈夫かな?

**タツヤ** 画像を読み込んだActorオブジェクトを並べて表示するだけだからね。簡単。

**レイ** センセ〜、前回の迷路のプログラムを思い出しながら、プレイヤーのエイリアンさんを動かすコードを書いたの〜。見てぇ。

carry_cargo.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

map_data =[[0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
           [1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 1, 1, 1, 1, 1, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
           [1, 0, 0, 1, 0, 1, 0, 0, 0, 1],
           [0, 0, 0, 0, 0, 1, 0, 1, 0, 0]]

player_location = [1, 0] # playerの現在位置[x, y]
exit_location = [9, 6] # 出口の位置
size = 70 # タイルの一辺
player = Actor('player', topleft=(size, 0)) # プレイヤー
floor = Actor('floor', topleft=(0, 0)) # 床のタイル
box = Actor('box', topleft=(0, 0)) # 箱
# 出口
exit = Actor('exit', topleft=(size*exit_location[0], size*
exit_location[1]))

def draw():
    screen.clear()
    for y in range(len(map_data)):
        for x in range(len(map_data[0])):
            floor.topleft=(size*x, size*y)
```

## carry_cargo.pyの続き

```
            floor.draw()
            if map_data[y][x] != 0:
                box.topleft=(size*x, size*y)
                box.draw()
    exit.draw()
    player.draw()

def on_key_down(key):
    if key == keys.UP:
        # プレイヤーが上端でなければ
        if player_location[1] > 0:
            # プレイヤーの進む方向がboxでなければ進む
            if map_data[player_location[1]-1][player_locat
ion[0]] != 1:
                player_location[1] -= 1
                player.y -= size

    elif key == keys.DOWN:
        # プレイヤーが下端でなければ
        if player_location[1] < len(map_data)-1:
            if map_data[player_location[1]+1][player_locat
ion[0]] != 1:
                player_location[1] += 1
                player.y += size

    elif key == keys.LEFT:
        # プレイヤーが左端でなければ
        if player_location[0] > 0 :
            if map_data[player_location[1]][player_locati
on[0]-1] != 1:
                player_location[0] -= 1
                player.x -= size

    elif key == keys.RIGHT:
        # プレイヤーが右端でなければ
        if player_location[0] < len(map_data[0])-1:
            if map_data[player_location[1]][player_locati
on[0]+1] != 1:
                player_location[0] += 1
                player.x += size

pgzrun.go()
```

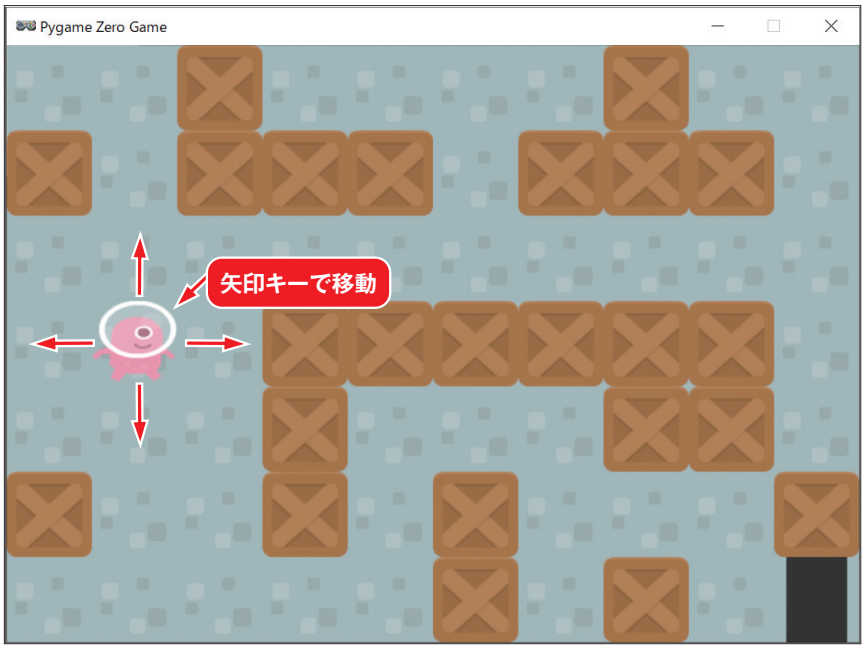
何かキーが押されると自動的に呼びだされるon_key_down関数

プレイヤーの進む方向にboxがなければプレイヤーを進めることができる

どのキーが押されたのかは引数keyを調べるとわかる

**タツヤ** い、いつの間に…。ちゃんと動く…（図1）。

**図1●上下左右の矢印キーでプレイヤーを動かせる**




**先生** ほぅ、しっかりコーディングできているね。on_key_down関数の使い方も間違っていない。座標計算や繰り返し処理も、数値を使わずに変数や関数を使って値を求めているところなんて、ホント素晴らしい。

**レイ** えへへ。

**タツヤ** で、でもさぁ、前回やった内容だしさ〜。僕なら、もっと「読みやすく」アップグレードできるけどね〜。

**先生** それでは、タツヤ君。レイちゃんのコードを参考にして、プレイヤーの進む方向に障害物の箱（box）がなければTrueを返す「is_not_box関数」を作ってくれないかな? 引数はon_key_down関数に渡される「key」と、player_locationを受け取る「location」だ。さらに、プレイヤーの進む方向に箱があるときは「False」を返すようにして欲しい。

**タツヤ** うぐぐぐ…マジか。

**レイ** や〜い。

**タツヤ** よ〜し、やってやる。is_not_box関数の宣言は「def is_not_box(key, location):」でいいな。処理は、もし「key == keys.UP」なら、プレイヤーの上に箱があるかどうかを調べて、なければTrueを返す、あったらFalseを返す…と。これを、keys.DOWN、keys.LEFT、keys.RIGHTでもやればいいはずだ…。こんな感じか?

### is_not_box関数

```
def is_not_box(key, location):
    if key == keys.UP:
        if map_data[location[1]-1][location[0]] != 1:
            return True
    elif key == keys.DOWN:
        if map_data[location[1]+1][location[0]] != 1:
            return True
    elif key == keys.LEFT:
        if map_data[location[1]][location[0]-1] != 1:
            return True
    elif key == keys.RIGHT:
        if map_data[location[1]][location[0]+1] != 1:
             return True
    return False
```

Actorの進む方向がboxでなければTrueを返す

**先生** そうだ。完璧じゃないか。それではレイちゃん。タツヤ君が作ったis_not_box関数をレイちゃんのコードに貼り付けよう。それから、先生がプレイヤーを1マス動かす「move関数」を作ったので、こちらも貼り付けて欲しい。

### move関数

```
def move(key, location, obj):
    if key == keys.UP:
        location[1] -= 1
        obj.y -= size
    elif key == keys.DOWN:
        location[1] += 1
        obj.y += size
    elif key == keys.LEFT:
        location[0] -= 1
        obj.x -= size
    elif key == keys.RIGHT:
        location[0] += 1
        obj.x += size
```

キーの方向にActorを進める

**レイ** わ〜い、関数が2つ増えた!

**先生** それでは、is_not_box関数とmove関数を使ってon_key_down関数を書き換えてみよう。「プレイヤーは上端にいない、かつ、プレイヤーの上はboxではない」といった判断を、「and演算子」を使って1行で記述しているよ。

### on_key_down関数

```
def on_key_down(key):
    if key == keys.UP:
        if player_location[1] > 0 and is_not_box(key, play
er_location):
            move(key, player_location, player)

    elif key == keys.DOWN:
        if player_location[1] < len(map_data)-1 and is_
not_box(key, player_location):
            move(key, player_location, player)

    elif key == keys.LEFT:
        if player_location[0] > 0 and is_not_box(key, play
er_location):
            move(key, player_location, player)

    elif key == keys.RIGHT:
        if player_location[0] < len(map_data[0])-1 and is_
not_box(key, player_location):
            move(key, player_location, player)
```

プレイヤーは上端にいない、かつ、プレイヤーの上はboxではない

プレイヤーは下端にいない、かつ、プレイヤーの下はboxではない

プレイヤーは左端にいない、かつ、プレイヤーの左側はboxではない

プレイヤーは右端にいない、かつ、プレイヤーの右側はboxではない

**レイ** すご〜い。on_key_down関数がすっごくわかりやすくなった。

**タツヤ** 協力しあうって…すばらしいなぁ。

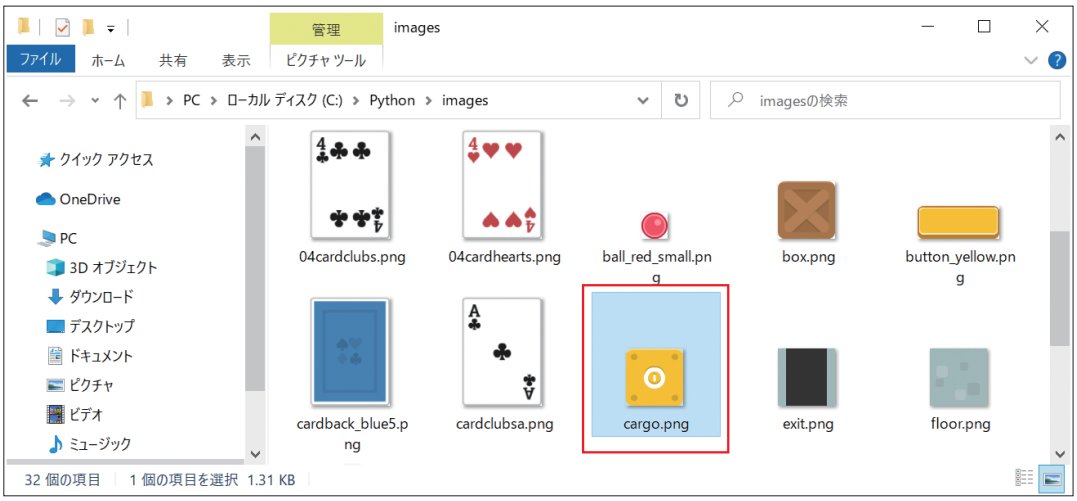
# 5日目 Part 2 荷物の表示と移動

**先生** 今度は、プレイヤーが運ぶ荷物を用意しよう。

**レイ** 荷物は、どの画像を使うの?

**先生** 今回は、「platformer-art-completepack-0」フォルダーの「Base pack」→「Tiles」にある「boxCoin.png」を使う。imagesフォルダーにコピーしたら、ファイル名を「cargo.png」に変えておこう(**図1**)。

**図1●プレイヤーが運ぶ荷物の画像**




**タツヤ** まあ、荷物に見えなくもない。

**先生** 荷物を表示する方法は、他のActorオブジェクトと同じだ。ただし、プレイヤーと同じように「荷物が今どこにあるのか?」を保持する「cargo_location」というグローバル変数を用意しよう。で、次のようにコードを変更して実行すれば、map_dataの「1, 1」の場所に荷物が表示されるはずだ。

carry_cargo.pyを変更

```
...

cargo_location = [1, 1]   # 荷物の現在位置

...

cargo = Actor('cargo', topleft=(size*1, size*1)) # 荷物

...

def draw():
    screen.clear()
    for y in range(len(map_data)):
        for x in range(len(map_data[0])):
            floor.topleft=(size*x, size*y)
            floor.draw()
            if map_data[y][x] != 0:
                box.topleft=(size*x, size*y)
                box.draw()
    exit.draw()
    player.draw()
    cargo.draw() # 荷物の表示
```

**レイ** わ、表示された（図2）。

**タツヤ** この荷物をプレイヤーが運ぶにはどうすればいいんだ?

**先生** プレイヤーが荷物を後ろから「押す」イメージだ。

**レイ** それじゃ、障害物の箱の配置やプレイヤーの位置によっては、押せない場所に荷物が来てしまうこともあるわけね。

図2●荷物を表示

**先生** そこが、このパズルゲームの「パズル」の部分だ。荷物が動かせなくなったら、ゲームオーバー。当然、運べる経路が少ない方が難しくなる。

**タツヤ** 面白そうだぞぅ。処理としては、プレイヤーが移動する先に荷物があれば、プレイヤーと一緒に荷物を同じ方向へ移動すればいいだけだね。

**レイ** でも、荷物の移動先が箱や外周の壁だったら移動できないわ。移動先に箱や壁がないか、調べてから動かさなきゃね。

**先生** それじゃ、タツヤ君は、プレイヤーが移動する先が荷物かどうかを調べる「is_cargo関数」を考えて欲しい。レイちゃんは、荷物が移動できるのなら荷物を移動してTrueを返す、移動できないのならFalseを返す「is_push関数」を作って欲しい。どちらも引数はkeyだけだ。

**タツヤ** 移動する先に荷物があるかどうか調べるなんて簡単。移動する方向はkeyを調べればわかるし、player_locationでプレイヤーの位置もわかる。プレイヤーの移動先に荷物があればTrueを返せばいいんでしょ…できましたぁ。

### is_cargo関数

```
def is_cargo(key):
    if key == keys.UP:
        if player_location[0] == cargo_location[0] and player_
location[1]-1 == cargo_location[1]:
            return True
    elif key == keys.DOWN:
        if player_location[0] == cargo_location[0] and player_
location[1]+1 == cargo_location[1]:
            return True
    elif key == keys.LEFT:
        if player_location[0]-1 == cargo_location[0] and play
er_location[1] == cargo_location[1]:
            return True
    elif key == keys.RIGHT:
        if player_location[0]+1 == cargo_location[0] and play
er_location[1] == cargo_location[1]:
            return True
    return False
```

プレイヤーの上に荷物があるときTrue

プレイヤーの下に荷物があるときTrue

プレイヤーの左側に荷物があるときTrue

プレイヤーの右側に荷物があるときTrue

**レイ** タツヤ、やるわね。センセ〜、助けてぇ。荷物が移動できるかどうかはどうやって調べるの〜。

**先生** プレイヤーと同じだ。移動先が外周の壁かどうかはcargo_locationで調べる。箱のboxがあるかどうかは、is_not_box関数を使えばいい。荷物の移動にはmove関数を使う。

**レイ** そうか。locationをプレイヤーじゃなくて荷物にすればいいのね。move関数は、荷物のActorオブジェクトを渡せば荷物が移動する…。これでいいのかしら。

### is_push関数

```
def is_push(key):
    if key == keys.UP:
        if cargo_location[1] > 0 and is_not_box(key, cargo_loc
ation):
            move(key, cargo_location, cargo)
            return True
    elif key == keys.DOWN:
        if cargo_location[1] < len(map_data)-1 and is_not_box(
key, cargo_location):
            move(key, cargo_location, cargo)
            return True
    elif key == keys.LEFT:
        if cargo_location[0] > 0 and is_not_box(key, cargo_loc
ation):
            move(key, cargo_location, cargo)
            return True
    elif key == keys.RIGHT:
        if cargo_location[0] < len(map_data[0])-1 and is_not_
box(key, cargo_location):
            move(key, cargo_location, cargo)
            return True
    return False
```

荷物は上端ではなく、かつ、荷物の上にboxはない

荷物は下端ではなく、かつ、荷物の下にboxはない

荷物は左端ではなく、かつ、荷物の左にboxはない

荷物は右端ではなく、かつ、荷物の右にboxはない

**先生** できたね。最後は、すべての関数を組み合わせて、プレイヤーが荷物を押して移動できるように、on_key_down関数を書き換えよう。例えば、「上矢印キー」が押されたら、プレイヤーの上は壁か箱か荷物かを調べ、荷物だった場合は荷物が上に移動できるかを調べ、移動できるのなら荷物を移動してプレイヤーも移動する。

## on_key_down関数

```
def on_key_down(key):
    if key == keys.UP:
        if player_location[1] > 0:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.DOWN:
        if player_location[1] < len(map_data)-1:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.LEFT:
        if player_location[0] > 0:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.RIGHT:
        if player_location[0] < len(map_data[0])-1:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key,player_location):
                move(key, player_location, player)
```

プレイヤーの進む方向に荷物があるなら、荷物が移動できるかどうか調べ、移動できたら、プレイヤーも一緒に移動する。荷物ではないなら、boxがあるかどうか調べ、なければプレイヤーを移動する

**レイ** 条件がマジヤバイ。

**先生** それから、障害物の箱の配置も、迷路のプログラムのものだと、荷物を出口まで運べないのでmap_dataを変更しよう。

### map_dataの変更

```
map_data =[[0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
           [1, 0, 1, 1, 0, 1, 0, 0, 1, 1],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 0, 1, 1, 0, 1, 1, 0, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
           [0, 1, 1, 1, 0, 0, 0, 0, 0, 0]]
```

プレイヤーが出口まで荷物を押していけるように障害物の箱（box）の配置を考える

**タツヤ** できたぁ。これで荷物が運べるぞ（図3）。

**図3●プレイヤーが荷物を出口まで運べるようになった**


**先生** これで、荷物運びパズルゲームの基本的な部分は完成だ。いよいよ、ゲームとしての最終仕上げといこう。

# 5日目 Part 3 荷物運びパズルゲームの完成

**先生** それでは、荷物運びパズルゲームに「ゲームオーバー」画面と「ステージクリア」画面を追加しよう。最初に考えるのは、ステージクリア画面だ。荷物が出口と同じ場所に来たときに、ステージクリア画面に切り替えよう。

**タツヤ** ステージをクリアしたら、次のステージに進むんじゃね?

**先生** 今回、ステージは1種類のみだ。もし、複数のステージを用意したいのなら、map_dataの種類を増やして、ステージクリア後に切り替えればいいだろう。

**タツヤ** ヨッシャ〜。10ステージくらい作ったるで。

**先生** 次に考えるのは、荷物が動かしようのない状態になったらゲームオーバー画面にすることだ。

**レイ** 荷物が動かないって、どんな状態?

**先生** 荷物は、プレイヤーが上下左右から押すことで移動する。つまり、荷物の隣り合う2つの辺が壁か箱に面していると、荷物は移動できなくなる。

**タツヤ** 荷物の縦と横の辺が壁とかにくっついていないか調べるのか…。

**レイ** あと、移動できなくなった途端にいきなり「ゲームオーバー」だと、何が起こったのかわかりにくいから、少し間をおいて表示するようにしないとね(**図1**)。

図1●ゲームオーバーとステージクリアの表示


**先生** それじゃ、最後に荷物運びパズルゲームの最終的なサンプルコードを紹介する。参考にしてコーディングしてみよう。

carry_cargo.py

```
import pgzrun

WIDTH = 700
HEIGHT = 490

map_data =[[0, 0, 1, 0, 0, 1, 0, 0, 0, 0],
           [1, 0, 1, 1, 0, 1, 0, 0, 1, 1],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [1, 0, 1, 1, 0, 1, 1, 0, 1, 0],
           [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
           [0, 1, 1, 1, 0, 0, 0, 0, 0, 0]]
```

## carry_cargo.pyの続き

```
player_location = [1, 0] # playerの現在位置[x, y]
cargo_location = [1, 1]  # 荷物の現在位置
exit_location = [9, 6] # 出口の位置
size = 70 # タイルの一辺
player = Actor('player', topleft=(size, 0)) # プレイヤー
floor = Actor('floor', topleft=(0, 0)) # 床のタイル
box = Actor('box', topleft=(0, 0)) # 箱
# 出口
exit = Actor('exit', topleft=(size*exit_location[0], size*
exit_location[1]))
cargo = Actor('cargo', topleft=(size*1, size*1)) # 荷物
status = 1 # 1:開始画面、2:終了、3:判定表示
message = '' # 判定結果メッセージ
time_count = 0 # 表示までのインターバル用

def draw():
    screen.clear()
    if status == 3:
        screen.draw.text(message, (200, 220), color='oran
ge', fontsize=72)
    else:
        for y in range(len(map_data)):
            for x in range(len(map_data[0])):
                floor.topleft=(size*x, size*y)
                floor.draw()
                if map_data[y][x] != 0:
                    box.topleft=(size*x, size*y)
                    box.draw()
        exit.draw()
        player.draw()
        cargo.draw()

def update():
    global status
    global time_count
    if status == 2:
        time_count += 1
        if time_count == 60:
            status = 3

def on_key_down(key):
    if key == keys.UP:
```

追加

追加

追加

## carry_cargo.pyの続き

```
        if player_location[1] > 0:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.DOWN:
        if player_location[1] < len(map_data)-1:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.LEFT:
        if player_location[0] > 0:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key, player_location):
                move(key, player_location, player)
    elif key == keys.RIGHT:
        if player_location[0] < len(map_data[0])-1:
            if is_cargo(key):
                if is_push(key):
                    move(key, player_location, player)
            elif is_not_box(key,player_location):
                move(key, player_location, player)
    judgment()

def is_not_box(key, location):
    if key == keys.UP:
        if map_data[location[1]-1][location[0]] != 1:
            return True
    elif key == keys.DOWN:
        if map_data[location[1]+1][location[0]] != 1:
            return True
    elif key == keys.LEFT:
        if map_data[location[1]][location[0]-1] != 1:
            return True
    elif key == keys.RIGHT:
        if map_data[location[1]][location[0]+1] != 1:
            return True
```

追加

## carry_cargo.pyの続き

```
    return False

def is_cargo(key):
    if key == keys.UP:
        if player_location[0] == cargo_location[0] and pla
yer_location[1]-1 == cargo_location[1]:
            return True
    elif key == keys.DOWN:
        if player_location[0] == cargo_location[0] and pla
yer_location[1]+1 == cargo_location[1]:
            return True
    elif key == keys.LEFT:
        if player_location[0]-1 == cargo_location[0] and
player_location[1] == cargo_location[1]:
            return True
    elif key == keys.RIGHT:
        if player_location[0]+1 == cargo_location[0] and
player_location[1] == cargo_location[1]:
            return True
    return False

def is_push(key):
    if key == keys.UP:
        if cargo_location[1] >= 1:
            if is_not_box(key, cargo_location):
                move(key, cargo_location, cargo)
                return True
    elif key == keys.DOWN:
        if cargo_location[1] <= (len(map_data)-2):
            if is_not_box(key, cargo_location):
                move(key, cargo_location, cargo)
                return True
    elif key == keys.LEFT:
        if cargo_location[0] >= 1:
            if is_not_box(key, cargo_location):
                move(key, cargo_location, cargo)
                return True
    elif key == keys.RIGHT:
         if cargo_location[0] <= (len(map_data[0])-2):
            if is_not_box(key, cargo_location):
                move(key, cargo_location, cargo)
                return True
```

## carry_cargo.pyの続き

```
    return False

def move(key, location, obj):
    if key == keys.UP:
        location[1] -= 1
        obj.y -= size
    elif key == keys.DOWN:
        location[1] += 1
        obj.y += size
    elif key == keys.LEFT:
        location[0] -= 1
        obj.x -= size
    elif key == keys.RIGHT:
        location[0] += 1
        obj.x += size

def judgment():
    global status
    global message
    if cargo_location == exit_location and status == 1:
        message = 'STAGE CLEAR'
        status = 2
    else:
        touch = 0
        x = cargo_location[0] # 荷物のx
        y = cargo_location[1] # 荷物のy
        if x == 0 or x == len(map_data[0])-1:
            touch += 1
        elif map_data[y][x-1] == 1 or map_data[y][x+1] ==
1:
            touch += 1
        if y == 0 or y == len(map_data)-1:
            touch += 1
        elif map_data[y-1][x] == 1 or map_data[y+1][x] ==
1:
            touch += 1
        if touch > 1 and status == 1:
            message = 'GAME OVER'
            status = 2

pgzrun.go()
```

追加

ゲーム中で、かつ、荷物と出口が重なったら
表示用のメッセージを作りstatusを「終了」にする

荷物の辺が壁や箱に触れている数

荷物の左右が壁か箱
に接している

荷物の上下が壁か箱
に接している

荷物の縦横が、両方壁や箱に接している場合、
荷物は移動できないのでゲームオーバー

**タツヤ** よ〜し、できた。あとは、ステージをドンドン用意して、さらにゲームを面白くしていくぞ〜。

**レイ** 私もできた〜。でも、センセ〜。先生のサンプルコードをもっと短く書けないか、いろいろやってみたんだけど、逆に小さな関数が増えたり、条件が横に長くなってゴチャゴチャ度が増しちゃうのよね。なんか、もっといい方法はないのかしら。

**タツヤ** それな〜。

**先生** レイちゃんの考え方は、とっても重要だ。複雑な処理を分岐処理だけで作ろうとすると、どんどん分岐が増えてコードが読みにくくなる。こうなると、どうやってコーディングをするかということより、プログラム全体の設計や構造が重要になってくる。

**レイ** プログラムの設計? 構造?

**先生** プログラムの設計方法はいろいろある。有名な方法は「構造化」や「オブジェクト指向設計」だ。ただ、アプリの設計については、もう少しプログラミングに慣れてきてからの方がいいだろう。今は、とにかく動くプログラムをたくさん作るといいね。5日間お疲れ様。また次の機会に会いましょう。それではオンライン切りま〜す。

**タツヤ** **レイ** センセ〜、楽しかったよ〜。バイバ〜イ。

■筆者紹介

中島 省吾

有限会社メディアプラネット代表取締役。テクニカルライターとして、ネットワークやプログラミング関連の記事を執筆するほか、IT企業向けのセミナーや新人研修の講師なども手掛ける。著書は「ビジネスPython超入門」（日経BP）など多数。

# 5日でできる！ Pythonゲームプログラミング入門

著者●中島 省吾
編集●日経ソフトウエア
発行人●中野 淳
編集長●久保田 浩
デザイン・制作●JMCインターナショナル

発行●日経BP
Nikkei Business Publications, Inc.
発売●日経BPマーケティング
〒105-8308 東京都港区虎ノ門4-3-12
URL●https://nkbp.jp/bpshopqa



日経BP