テトリス on Pygame 前編

Swimmy 高田馬場校

2024 年 8 月 29 日

目次

第 1 章　はじめに 5

第 2 章　今まで通りに作る 7

2.1　いつも通り作ってみよう 7

第 3 章　オブジェクト指向プログラミングへ 11

3.1　オブジェクト指向プログラミング 11

3.2　クラスの作り方 12

3.3　クラスの中に変数を作る 14

3.4　クラスの中に関数を作る 17

3.5　まとめ 18

第 4 章　テトリスでクラスを使おう 19

4.1　main 関数の役割を分担する 19

4.2　Board クラスを定義する 19

4.3　main 関数を書き換える 20

4.4　クラスの設計を練習する 26

第 5 章　テトリスのカーソルを作る 33

5.1　カーソルの設計 33

5.2　Cursor クラスを定義する 35

5.3　カーソルを動かす 37

5.4　まとめ 40

第 1 章

はじめに

わからないことがあっても気にしないでください。これから学ぶオブジェクト指向は本来大学の専門科目にあたります。今も活発に研究が行われている分野ですし、今後 10 年で大きく変わる可能性もあります。この本でおすすめしていることも、将来的にはアンチパターン(やってはいけない設計)と言われているかもしれません。今自分はプログラミング言語の最前線を学んでいるんだなあと思って楽しんでくれれば嬉しいです。

第 2 章

今まで通りに作る

2.1　いつも通り作ってみよう

ヘビゲームと同じように、今までのように作ってみましょう。

今までのような作り方

```
# tetris using pygame
import pygame

# 定数(決まっている数)の定義
# テトリスの盤面は横 10マス、縦 22マス
WIDTH = 10
HEIGHT = 22

# 1マスの大きさをピクセルで表す
TILE_SIZE = 30

# 色の定義
# コンピュータは色を赤、緑、青の強さを 3つの値として表します
# 光は全色混ぜると白になります
WHITE = (255, 255, 255)

# 光っていないと黒です
BLACK = (0, 0, 0)

# 白と黒の間、すなわち灰色です
GRAY = (128, 128, 128)

# どんな色になるか考えてみましょう
```

```
24 LIGHT_GRAY = (200, 200, 200)
25 CYAN = (0, 255, 255)
26 BLUE = (0, 0, 255)
27 ORANGE = (255, 165, 0)
28 YELLOW = (255, 255, 0)
29 GREEN = (0, 128, 0)
30 PURPLE = (128, 0, 128)
31 RED = (255, 0, 0)
32
33 def main():
34     # Create the board
35     """
36     盤面と対応するリストを作ります
37     盤面は幅１０高さ２２なので
38     「黒を１０個並べたもの」を２２個並べたリストを作ります
39     下の表記はリスト内包表記と呼ばれるものです
40     [0 for _ in range(10)] は [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] と同
           じです.
41
42     今回は最初の状態として全てのマスを黒にしています。
43     """
44     board = [[BLACK for _ in range(WIDTH)] for _ in range(HEIGHT)]
45
46     # なぜ呼ぶのかは考えなくてOK
47     pygame.init()
48
49     # 画面の解像度を設定して作成
50     # この行があって初めてウィンドウが作成される
51     screen = pygame.display.set_mode((WIDTH * TILE_SIZE, HEIGHT *
           TILE_SIZE))
52
53     # ウィンドウのタイトルを設定
54     pygame.display.set_caption("Tetris")
55
56     # ゲーム内の時刻を表す変数になります
57     clock = pygame.time.Clock()
58
59     while True:
60         # 画面を表示します
61         # 縦にHEIGHT(22)回、
```

```
62          for y in range(HEIGHT):
63              # 横にWIDTH(10)回、
64              for x in range(WIDTH):
65                  # それぞれのマスを描画します
66                  pygame.draw.rect(screen, board[y][x], (x * TILE_SIZE,
                        y * TILE_SIZE, TILE_SIZE, TILE_SIZE))
67
68          # 線を描画します
69          # 縦線を 22本
70          for y in range(HEIGHT):
71              pygame.draw.line(screen, GRAY, (0, y * TILE_SIZE), (WIDTH
                    * TILE_SIZE, y * TILE_SIZE))
72          # 横線を 10本
73          for x in range(WIDTH):
74              pygame.draw.line(screen, GRAY, (x * TILE_SIZE, 0), (x *
                    TILE_SIZE, HEIGHT * TILE_SIZE))
75
76          # 画面が消されたらmain も return で終了します
77          for event in pygame.event.get():
78              if event.type == pygame.QUIT:
79                  return
80
81          # この行で描画内容が実際のモニターに反映されます
82          pygame.display.update()
83
84          # 画面の更新頻度を設定します。この場合は一秒間に 60回更新します
85          clock.tick(60) # 60fps
86
87  if __name__ == "__main__":
88      main()
```

実行結果

ここまでできたら、次の章に進んでください。実行結果も載せてあるので比べてみてください。

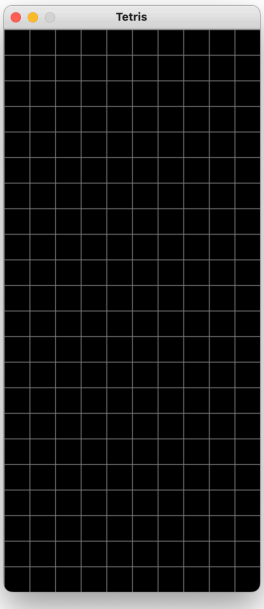

図 2.1　テトリスの実行結果

第 3 章

オブジェクト指向プログラミングへ

3.1　オブジェクト指向プログラミング

プログラミング言語界の主人公的存在、オブジェクト指向

今まで私たちは「変数」、「関数」を用いてプログラムを作ってきました。

ソースコード 3.1　復習

```
1 # 変数の作り方
2 名前 = "ATAI"
3
4 # 関数の作り方
5 def 関数名():
6     # その時にすることを書く
7     print("こんにちはよいお日和ですね")
```

変数はデータに名前をつけることで、関数は処理に名前をつけることでした[*1]。オブジェクト指向プログラミングは主に「クラス」という「プログラムそのものに名前をつけたもの」を構成することでプログラムを作ります。クラスは「変数」と「関数」を持つことができます。つまり、どんな Python ファイルも一つのクラスにまとめることができます。

[*1] 厳密には、変数とはデータの領域に名前をつけることです。

3.2　クラスの作り方

3.2.1　クラスの作り方

クラスは以下のように作ります。Python ファイルの中で何個でも作ることができます[*2]。

[*2] しかし、みやすさの観点からクラスは 1 ファイルにつき 2〜3 個までにする人もいます。

ソースコード 3.2　クラスの作り方

```
1 class クラス名:
2     ...
```

クラス名は作りたいクラスに応じて変えてください。... は省略しているわけではなく、「後で書きます」というちゃんとした Python のプログラムです。

3.2.2　クラスを変数に入れる

クラスは設計図のようなもので、実際に使うには**作って変数に入れる**必要があります[*3]。

ソースコード 3.3　クラスを変数に入れる

```
1 変数名 = クラス名()
```

「クラスを作る」とは「クラス名 ()」と書くことです。作ったクラスは変数に入れないと次の行には捨てられています[*4]。なので、大体の場合は「変数名=クラス名 ()」と書くことが多いです。カッコの中に何が入るかは、クラスによって異なります。後で説明します。

関数と似てる？

クラスは関数と似ている部分があります。関数は使う前になるべく関係ないところで「def 関数名 ():」と書いて「宣言」[*5]しましたね。

ソースコード 3.4　関数の作り方

```
1 def 関数名():
2   # 呼ばれた時にすることを書く
3   ...
```

実際に使うときは、その場所で「関数名 ()」と書きましたね。

ソースコード 3.5　関数の呼び出し方

```
1 関数名() #「呼ばれた時にすること」が実行される
```

カッコの中には何かが入ったり入らなかったりしますが、関数を使うときは「関数名 ()」と書きました。似てますよね。プログラムが複雑になったときに宣言と使う場所を分離することでプログラムをみやすくするという考え方です。

[*3] このような変数のことをインスタンスと呼んだりします。Python の場合、全ての変数は何かのインスタンスになるらしいです。

[*4] 捨てられていないこともあります。Python は ARC と世代別 GC の二種類を持っているらしく、クラスに循環参照がある場合は GC により時間差で捨てられます

[*5] こういうものだよ、と決めること

3.3　クラスの中に変数を作る

3.3.1　クラスの中に変数を作る

普通の変数の作り方と**ほぼ**同じです。

ソースコード 3.6　クラスの中に変数を作る

```
1 class クラス名:
2     def __init__(self):
3         self.変数名 = 値
```

__init__関数は**クラスを作って変数に入れるときに自動で呼ばれます**。変数を作って入れることもできますし、print を書くとクラスをつくるたびに表示されるようにすることもできます。たとえば、教科書に「Mentor クラスを作って、中に age という変数を作る。age には最初に 20 を入れる。」という文章があったら、

ソースコード 3.7　Mentor クラスの作り方

```
1 class Mentor:
2     def __init__(self):
3         self.age = 20
```

こう書きましょう。**変数は一つだけでなく何個でも作れます**[*6]。さらに、「name という変数を作って、先生の名前を入れてみよう」という文章があったら、

ソースコード 3.8　Mentor クラスの作り方

```
1 class Mentor:
2     def __init__(self):
3         self.age = 20
4         self.name = "先生の名前を入れる"
```

クラスはたくさん書くことで設計の仕方がわかっていくので、何度も書く→増やす→失敗する→改良するを繰り返して慣れていきましょう。

3.3.2　クラスの変数の中身がわからない場合

でも、先生の年齢がいつも 20 歳なのはちょっと変ですね。先生が常に 20 歳とは限りません。**クラスは設計図**なので設計段階ではわからない数値やデータもあります。変数に入

[*6] パソコンの覚えられる量を超えない限り

れる値がまだわからない時は、引数にすれば**実際に作る時まで先延ばし**にすることができます。

ソースコード 3.9　引数を使う場合

```
1 class Mentor:
2     def __init__(self, age, name):
3         self.age = age
4         self.name = name
```

このように書くことで、Mentor クラスを**作る時に年齢と名前を指定**することができます。

ソースコード 3.10　Mentor クラスの作り方

```
1 # クラスを作る時に、年齢と名前を指定する
2 mentor1 = Mentor(20, "A 先生")
```

30 歳の先生を作る場合は、

ソースコード 3.11　Mentor クラスの作り方

```
1 # クラスを作る時に、年齢と名前を指定する
2 mentor2 = Mentor(30, "B 先生")
```

さっきのプログラム 2 つでは違う変数に同じクラスを作っていましたが、**クラスは設計図**なので、クラスを一度作っておくと、別のデータで別の変数に似たデータを入れられます。設計図を元にカスタマイズしながら量産できるということです。まだ便利さがわからないかもしれませんが、関数の引数とオブジェクト指向は便利さがわかりづらいランキング 1 位と 2 位なのでここは少し我慢してください[*7]。

3.3.3　クラスの変数の中身を変更する

クラスの変数の中身を変更するには、以下のように書きます。

ソースコード 3.12　クラスの変数の中身を変更する

```
1   mentor3 = Mentor(40, "C 先生")
2   mentor3.age = mentor3.age + 1
```

このように書くことで、mentor3 の年齢を 1 歳増やすことができます。お誕生日おめでとうございます。

[*7] 作者の個人の感想です

3.3.4　クラスの変数を使う

クラスの変数を使うには、以下のように書きます。

ソースコード 3.13　クラスの変数を使う

```
1  print(mentor3.age)
```

このように書くことで、mentor3 の年齢を表示することができます[*8]。普通の変数と同じように使えますね。変数の集まりがクラスと考える人たちもいます[*9]。

[*8] このようなインスタンスに対して「.」を使ってアクセスする変数をインスタンス変数といいます。

[*9] 厳密には変数を一つに集める機能はストラクチャというものにもあるので、「クラス＝変数の集まり」と覚えるのは不正確かもしれません

3.4　クラスの中に関数を作る

関数の作り方も普通の関数の作り方と**ほぼ**同じです。

ソースコード 3.14　クラスの中に関数を作る

```
1 class クラス名:
2     def 関数名(self):
3         ...
```

引数に self というものが入っています。これは実際に使うときには入れないので、「self, 欲しい引数（なければ self だけ）」と書くと覚えておきましょう[*10]。たとえば、教科書に「Car クラスを作って、run という関数を作る。run 関数は「走ります」と表示する」という文章があったら、

ソースコード 3.15　Car クラスの作り方

```
1 class Car:
2     def run(self):
3         print("走ります")
```

こう書きましょう。**関数も何個でも作れます**し、引数を使うこともできます。

ソースコード 3.16　引数を使う場合

```
1 class Car:
2     def run(self):
3         print("走ります")
4     def set_speed(self, speed):
5         print("時速", speed, "km で走ります")
```

テキストを読んでいて「クラスの中に〜」という文章があってよくわからない場合はこの章に戻ってきてください。ちなみに Car クラスの run 関数を使うには、

ソースコード 3.17　Car クラスの使い方

```
1 car = Car()
2 car.run()
```

と書きます。**クラスは設計図**なので、一度変数にしないと使えません。run 関数は引数 self があったので「run(何か)」としなければならない気もしますが、実際に使うときは「car.run()」と書くだけで大丈夫です。[*11]

[*10] self を書かなくても良い言語もありますし、作者個人はそうすべきだと思うのですが…

[*11] このように、インスタンスに「.」をつけて呼ぶことのできる関数をインスタンスメソッドと呼びます。

3.5　まとめ

クラスは「変数」と「関数」[*12]を中に持つことができます。クラスは設計図のようなもので、実際に使うには変数に入れる必要があります。変数に入れるときは「変数名=クラス名 ()」のようにしますが、クラスによってはカッコの中に何かを入れる必要があります。

先生と調べよう

答えがないものもあります。暇な時に調べてみてください。

- オブジェクト指向プログラミングの良いところと悪いところは？
- Python の__init__関数などにある self（自分自身）って何？
- Python のクラスに名前をつけるときのルールは？

[*12] 正確にはメソッドと呼ばれます。メソッド＝手順、手続き

第 4 章

テトリスでクラスを使おう

4.1 main 関数の役割を分担する

クラスを設計するときは、まずクラスの役割を決めます。ひとまず、**盤面を管理する**クラスを作ってみましょう。どんな機能、変数を持っているかはこちらで決めました[*1]。

- 盤面のブロックサイズを表す変数
- 盤面のブロックの状態を表すリスト
- 盤面をスクリーンに描画する関数
- 盤面のブロックサイズから画面の大きさを計算する関数

内容が決まってきたらクラスを作ります。

4.2 Board クラスを定義する

今回はクラスを別ファイルに書いて、インポートする方法にしてみます。建築で functions.py と main.py に分かれていたのと同じような感じです。まずはこれだけ作ってみましょう。ファイル名は「tetris.py」とし、以下のように書いてください。

teris.py

```
1 # tetris using pygame
2 import pygame
3
4 WIDTH = 10
5 HEIGHT = 22
6 TILE_SIZE = 30
```

[*1] 将来は自分で設計することになります。うまくできないと怒られます。

```
7
8  WHITE = (255, 255, 255)
9  BLACK = (0, 0, 0)
10 GRAY = (128, 128, 128)
11 CYAN = (0, 255, 255)
12 BLUE = (0, 0, 255)
13 ORANGE = (255, 165, 0)
14 YELLOW = (255, 255, 0)
15 GREEN = (0, 128, 0)
16 PURPLE = (128, 0, 128)
17 RED = (255, 0, 0)
18
19 class Board:
20     def __init__(self):
21         ...
22     def draw(self, screen):
23         ...
24     def window_size(self):
25         ...
```

draw 関数が screen を引数にとっているのは、建築で mc を引数にとっていたのと似ています。Board クラスは変数に screen を持っていないので、画面に書くときは一度**画面を借りる**必要があります。試しに 4 行目の「, screen」を消して実行してみてください。「screen is undefined」みたいなエラーが出るはずです[*2]。

4.3　main 関数を書き換える

4.3.1　Board クラスを使って main 関数を書く

ここまで読んだら、main.py を作ります。

main.py

```
1 import pygame
2 from tetris2 import Board
3
4 def main():
5     # 作ったクラスを変数に入れてみます
6     board = Board()
```

[*2] ちなみにこの段階ではでないです。試してくれた方はごめんなさい。

```
7
8       # Initialize pygame
9       pygame.init()
10
11      # ウィンドウのサイズはboard の window_size()メソッドで取得できることにします
12      # def window_size(self): の部分に対応しています
13      screen = pygame.display.set_mode(board.window_size())
14
15      pygame.display.set_caption("Tetris")
16      clock = pygame.time.Clock()
17
18      while True:
19          # ボードを描画
20          # def draw(self, screen): の部分に対応しています
21          board.draw(screen)
22
23          # Handle events
24          for event in pygame.event.get():
25              if event.type == pygame.QUIT:
26                  return
27          # Update the display
28          pygame.display.update()
29
30          # Tick
31          clock.tick(60)
32
33  if __name__ == "__main__":
34      main()
```

main 関数が少し短くなったと思います。main 関数が今までやっていた「盤面に合わせてブロックを書く→線を引く」という処理を Board クラス（とそれが入った変数 board）に分担したからです[*3]。でも、この状態ではまだ動きません。tetris.py のクラスの中で作った関数がまだ「...」のまま残っていますね。実装しましょう。

4.3.2 Board クラスを実装する

[*3] ここが大事なのでテキストを読んでいない人はこの文を探してみてね

tetris.py を完成させる

```
# tetris using pygame
import pygame

# Constants
WIDTH = 10
HEIGHT = 22
TILE_SIZE = 30

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GRAY = (128, 128, 128)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
ORANGE = (255, 165, 0)
YELLOW = (255, 255, 0)
GREEN = (0, 128, 0)
PURPLE = (128, 0, 128)
RED = (255, 0, 0)

class Board:
    def __init__(self):
        # board という変数を作って、全て黒のマスで埋める
        self.board = [[BLACK for _ in range(WIDTH)] for _ in range(
            HEIGHT)]
        # TILE_SIZE という変数を作って、30 を入れる
        self.TILE_SIZE = 30
    def draw(self, screen):
        # 全マスに対して
        for y in range(HEIGHT):
            for x in range(WIDTH):
                # マスの色を描画
                pygame.draw.rect(screen, self.board[y][x], (x * self.
                    TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE,
                    self.TILE_SIZE))

        # 横線を描画
        for y in range(HEIGHT):
```

```
36              pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (
                    WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
37          # 縦線を描画
38          for x in range(WIDTH):
39              pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x
                     * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))
40      def window_size(self):
41          # ウィンドウのサイズを計算します
42          # 1マスがTILE_SIZE に入っていて、
43          # それが横幅はWIDTH 個、縦幅は HEIGHT 個あるので
44          # WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE がウィンドウのサイズ
                になります
45          return (WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE)
```

実行すると、一章と同じように動くはずです。

4.3.3 ブロックのサイズを変更できるようにする

ついでに、ブロックのサイズを main 関数から変更できるようにしましょう。

tetris.py を改造する

```
1  # tetris using pygame
2  import pygame
3
4  # Constants
5  WIDTH = 10
6  HEIGHT = 22
7  TILE_SIZE = 30
8
9  # Colors
10 WHITE = (255, 255, 255)
11 BLACK = (0, 0, 0)
12 GRAY = (128, 128, 128)
13 CYAN = (0, 255, 255)
14 BLUE = (0, 0, 255)
15 ORANGE = (255, 165, 0)
16 YELLOW = (255, 255, 0)
17 GREEN = (0, 128, 0)
18 PURPLE = (128, 0, 128)
19 RED = (255, 0, 0)
```

```
20
21 class Board:
22     def __init__(self, tile_size):
23         # 作る時にタイルサイズを指定する
24         self.board = [[BLACK for _ in range(WIDTH)] for _ in range(
               HEIGHT)]
25         self.TILE_SIZE = tile_size
26     def draw(self, screen):
27         for y in range(HEIGHT):
28             for x in range(WIDTH):
29                 pygame.draw.rect(screen, self.board[y][x], (x * self.
                       TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE,
                       self.TILE_SIZE))
30
31         # Draw the grid
32         for y in range(HEIGHT):
33             pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (
                   WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
34         for x in range(WIDTH):
35             pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x
                    * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))
36     def window_size(self):
37         return (WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE)
```

main.py を改造する

```
1 import pygame
2 from tetris import Board
3
4 def main():
5     # Create the board
6     board = Board(tile_size=30)
7
8     # Initialize pygame
9     pygame.init()
10    screen = pygame.display.set_mode(board.window_size())
11    pygame.display.set_caption("Tetris")
12    clock = pygame.time.Clock()
13
14    while True:
15        # Draw the board
```

```
16          board.draw(screen)
17
18          # Handle events
19          for event in pygame.event.get():
20              if event.type == pygame.QUIT:
21                  return
22          # Update the display
23          pygame.display.update()
24
25          # Tick
26          clock.tick(60)
27
28  if __name__ == "__main__":
29      main()
```

「board = Board(tile_size=30)」を変えることで、一マスのサイズを変更できます。ここまでできたら完璧です。

先生と考えよう

どちらも正解がないので、暇な時に考えてみてください。

- Board クラスの中に screen を作らなかったのはなぜ？
- main 関数の残された役割は？

4.4　クラスの設計を練習する

4.4.1　練習問題 1

まずは以下のプログラムを動かしてみましょう。その後、クラスを使って書き換えてみてください。

練習問題 1

```
account1_name = "Alice"
account1_balance = 100

account2_name = "Bob"
account2_balance = 50

while True:
    user_name = input("名前をどうぞ:␣")
    command = input("+␣or␣-:␣")
    amount = int(input("いくらにしますか:␣"))
    if user_name == account1_name:
        if command == "+":
            account1_balance += amount
        elif command == "-":
            if account1_balance < amount:
                print("残高が足りません")
                continue
            account1_balance -= amount
    elif user_name == account2_name:
        if command == "+":
            account2_balance += amount
        elif command == "-":
            if account2_balance < amount:
                print("残高が足りません")
                continue
            account2_balance -= amount
    print(f"{account1_name}:␣{account1_balance}")
    print(f"{account2_name}:␣{account2_balance}")
```

この while True: では、入力、コマンド確認、残高確認、出力の機能がすべて混ざっています。どういうふうに分離するといいでしょうか？

例

練習問題 1 の解答例

```
class Account: # Account = 口座
    def __init__(self, name, balance):
        self.name = name
        self.balance = balance # balance = 残高

    def deposit(self, amount): # deposit = 預ける
        self.balance += amount

    def withdraw(self, amount): # withdraw = 引き出す
        if amount > self.balance:
            print("残高が足りません")
            return
        self.balance -= amount # amount = 金額

    def show(self):
        print(f"{self.name}: {self.balance}")

account1 = Account("Alice", 100)
account2 = Account("Bob", 50)

while True:
    user_name = input("名前をどうぞ: ")
    command = input("+ or - : ")
    amount = int(input("いくらにしますか: "))
    if user_name == account1.name:
        if command == "+":
            account1.deposit(amount)
        elif command == "-":
            account1.withdraw(amount)
    elif user_name == account2.name:
        if command == "+":
            account2.deposit(amount)
        elif command == "-":
            account2.withdraw(amount)
    account1.show()
    account2.show()
```

残高が足りているか確認するのは口座の役割に分担しました。また、残高と名前は別の変数でなく、一つのクラスにまとめました。同じものに関する情報（変数）は一つのクラスにまとめるのがおすすめです。

4.4.2　練習問題 2

まずは以下のプログラムを動かしてみましょう。ゲームになっているので先生と一緒に遊んでみてください。原因が分かり次第修正します。

練習問題 2

```
import os
player1_name = input("プレイヤー 1の名前を入力してください:␣")
player2_name = input("プレイヤー 2の名前を入力してください:␣")

words = []

# 最初に単語を入力するのはplayer1
words.append(input(f"{player1_name}は最初の単語を入力してください:␣"))

# 最初に答えるのはplayer2
player = player2_name
while True:
    for i in range(len(words)):
        data = input(f"{player}は{i+1}番目の単語を入力してください:␣")
        if data != words[i]:
            print(f"{player}の負けです")
            # 終了
            exit()
        else:
            print("正解です,␣つづけてください")
    # 出力を消す
    os.system("clear")
    print("ターンクリア、新しく単語を入力してください")
    words.append(input(f"{player}は新しく覚える単語を入力してください:␣"))

    # プレイヤーを交代する
    if player == player1_name:
        player = player2_name
    else:
        player = player1_name
```

クラスに分けるとどうなるでしょうか。作るクラスが一つとは限りません[*4]。

[*4] 2 つ以上が正解と言いたいわけではありません。メンターさんは学校の先生と違ってプロじゃないので顔から考えていることがバレやすいです。でも、心を読まれて答えを見つけられると微妙な気持ちになります。担当のメンターさんも経験しているかもしれません。

例

練習問題 2 の解答例

```
import os

class Player:
    def __init__(self):
        self.name = input("プレイヤーの名前を入力してください: ")

class MemoryGame:
    def __init__(self, player1, player2):
        self.player1 = player1
        self.player2 = player2
        self.words = []

        # ゲームオーバーかどうか
        self.game_over = False

        # 最初に単語を入力するのはplayer1
        self.words.append(input(f"{self.player1.name}は最初の単語を入力してください: "))

        # 最初に答えるのはplayer2
        self.current_player = self.player2 # current = 現在の

    def play(self):
        for word in self.words:
            data = input(f"{self.current_player.name}は単語を入力してください: ")
            if data != word:
                print(f"{self.current_player.name}の負けです")
                self.game_over = True
                return
            else:
                print("正解です, つづけてください")
        # 出力を消す
        os.system("clear")
        print("ターンクリア、新しく単語を入力してください")
```

```
34          self.words.append(input(f"{self.current_player.name}は新しく覚え
                る単語を入力してください:␣"))
35
36          # プレイヤーを交代する
37          if self.current_player.name == self.player1.name:
38              self.current_player = self.player2
39          else:
40              self.current_player = self.player1
41
42  player1 = Player()
43  player2 = Player()
44  game = MemoryGame(player1, player2)
45  while True:
46      game.play()
47      if game.game_over:
48          break
```

クラスは複雑になりますが、42〜48 行目は短く、分かりやすくなっています。Player() とするだけで入力欄が開くのは便利ですね[*5]。

[*5] でも、設計的にはダメな場合もありますので安易にこうしてはいけません

第 5 章

テトリスのカーソルを作る

5.1　カーソルの設計

5.1.1　カーソルの機能を考える

　カーソルというと、パソコンの矢印のことを思い浮かべるかもしれませんが、現在の場所を示すものを指します。今回は、テトリスのカーソルを作ります。カーソルはどんな変数・関数を持つといいでしょうか？今回も教科書の方で決めさせてもらいました。

- カーソルの位置を表す変数 2 つ
- カーソルを上下左右に動かす関数
- カーソルを描画する関数

今回はカーソルのある列は灰色にすることにします。また、カーソルが盤面からはみ出さないようにする機能も上下左右に動かす関数が持つことにします[*1]。

5.1.2　カーソルの設計について考える

「カーソルを描画する関数」とありますが、カーソルを描画する方法は 2 つあります。

- カーソルが screen に対して描画する
- カーソルが Board に指令を出し、Board が screen に描画する

これらの方法について考えてみましょう。どちらがいいでしょうか？

[*1] こういう処理を main 関数とかにやらせてしまうと「良くない設計」と言われかねません。プログラマの中には「間違った設計」を見るとすごい勢いで設計について語り出す熱心な人もいます。そうした人にも一目置かれるようなプログラマを目指したいですね。

5.1.3　設計にけりをつける

カーソルが screen に対して描画する派の主張

- カーソルは Board とは別の要素なので別に仕事を行うべき
- カーソルの情報を盤面に書き込むには、カーソルが盤面の情報にアクセスする必要があり、設計が増える

1 つ目の要素については個人の自由なのでどちらでもいいですが、2 つ目の要素は考慮する必要があります。

カーソルが Board に指令を出し、Board が screen に描画する派の主張

- カーソルも Board の一部
- screen に描画するクラスは一つにまとめた方がいい。Cursor も screen にアクセスするべきではない。

実際、カーソルが盤面の一部かどうかについては、**人によります**。でも、2 つ目の要素は重要です。screen に描画するクラスは一つにまとめた方がいいです。screen に関してバグがあった時に、screen にアクセスするクラスが一つにまとまっているとバグの原因を特定しやすくなります。

5.1.4　決着

今回は、このようにします。

- カーソルは Board の一部
- カーソルは描画を**行わない**

なので、それぞれのクラスの中身を以下のように変更します。

Board クラス

- 盤面のブロックサイズを表す変数
- 盤面のブロックの状態を表すリスト
- カーソルを表す変数
- 盤面, **カーソル**をスクリーンに描画する関数
- 盤面のブロックサイズから画面の大きさを計算する関数

Cursor クラス

- カーソルの位置を表す変数２つ
- カーソルを上下左右に動かす関数

5.2 Cursor クラスを定義する

5.2.1 Cursor クラスを定義する

tetris.py に Cursor クラスを追加します。初期状態は中央上側にカーソルがある必要があるので、__init__関数で初期位置を設定します。

Cursor クラスを作る

```
1  # tetris using pygame
2  import pygame
3
4  # Constants
5  WIDTH = 10
6  HEIGHT = 22
7  TILE_SIZE = 30
8
9  # Colors
10 WHITE = (255, 255, 255)
11 BLACK = (0, 0, 0)
12 GRAY = (128, 128, 128)
13 CURSOR_COLOR = (60, 60, 60)
14 CYAN = (0, 255, 255)
15 BLUE = (0, 0, 255)
16 ORANGE = (255, 165, 0)
17 YELLOW = (255, 255, 0)
18 GREEN = (0, 128, 0)
19 PURPLE = (128, 0, 128)
20 RED = (255, 0, 0)
21
22 class Board:
23     def __init__(self, tile_size):
24         # 作る時にタイルサイズを指定する
25         self.board = [[BLACK for _ in range(WIDTH)] for _ in range(
               HEIGHT)]
```

```
26          self.TILE_SIZE = tile_size
27
28          # Cursor
29          self.cursor = Cursor()
30      def draw(self, screen):
31          for y in range(HEIGHT):
32              for x in range(WIDTH):
33                  pygame.draw.rect(screen, self.board[y][x], (x * self.
                        TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE,
                        self.TILE_SIZE))
34
35          # Draw the cursor
36          for y in range(HEIGHT):
37              if self.board[y][self.cursor.x] == BLACK:
38                  pygame.draw.rect(screen, CURSOR_COLOR, (self.cursor.x *
                         self.TILE_SIZE, y * self.TILE_SIZE, self.
                        TILE_SIZE, self.TILE_SIZE))
39
40          # Draw the grid
41          for y in range(HEIGHT):
42              pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (
                    WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
43          for x in range(WIDTH):
44              pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x
                     * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))
45
46      def window_size(self):
47          return (WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE)
48
49  class Cursor:
50      def __init__(self):
51          self.x = WIDTH // 2
52          self.y = 0
53      def move_left(self):
54          self.x = max(0, self.x - 1)
55      def move_right(self):
56          self.x = min(WIDTH - 1, self.x + 1)
57      def move_down(self):
58          self.y = min(HEIGHT - 1, self.y + 1)
59      def move_up(self):
```

```
60          self.y = max(0, self.y - 1)
```

これだと Board に直接変数を追加しても良さそうな気もしますが、別の機能は別のクラスに書く、を徹底しましょう。Board クラスの__init__関数に「self.cursor = Cursor()」を追加していること、draw 関数の最後に「カーソルの列に対し、BLACK のマスを GRAY にする」処理を入れていることに注意してください。main 関数は変更しなくて大丈夫です。処理を任せることで変更部分を減らすことができるのもオブジェクト指向の特徴です。実行するとカーソルが出るはずです。

5.3 カーソルを動かす

5.3.1 キー入力を受け取る

カーソルを動かすには、キー入力を受け取る必要があります。さて、画面の話でもありましたが、キー入力を受け取る存在も一つにまとめた方がいいです。どうやって設計すればいいでしょうか？

- キー入力を受け取るクラスを作る
- main 関数にキー入力を受け取る処理を書く
- Board にキー入力を受け取る処理を書く
- Cursor にキー入力を受け取る処理を書く

このような方法が浮かんできたら、あなたもオブジェクト指向プログラマーの仲間入りです。

キーを受け取るクラスを作る方法はとても「アリ」なんですが、今回はキーの数が少ないのでしません。Board にキー入力を受け取る処理を書くと、Board がキー入力を受け取るという役割が増えてしまいますし、そもそも盤面に対してキーを打っているわけではないので、これは残念ながら良くないデザインとされます。Cursor も同様です。今回は「キー入力を受け取るのは main 関数の仕事」とします。後々一時停止や終了などのキーを作った時に、それをカーソルが受け取るのは不自然ですよね。

5.3.2 main 関数でキー入力を受け取る

先ほど決めた通り、main 関数でキー入力を受け取ることにします。

main 関数でキー入力を受け取る

```
1 import pygame
2 from tetris import Board
```

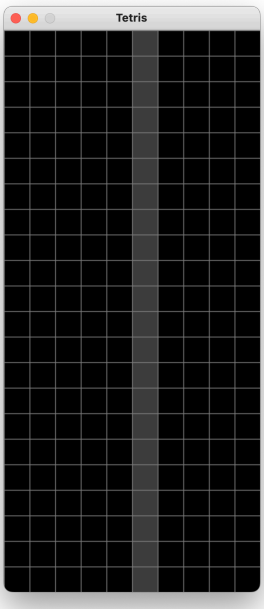

図 5.1　テトリスの実行結果

```
3
4 def main():
5     # Create the board
6     board = Board(tile_size=30)
7
8     # Initialize pygame
```

```
9       pygame.init()
10      screen = pygame.display.set_mode(board.window_size())
11      pygame.display.set_caption("Tetris")
12      clock = pygame.time.Clock()
13
14      while True:
15          # Draw the board
16          board.draw(screen)
17
18          # Handle events
19          for event in pygame.event.get():
20              if event.type == pygame.QUIT:
21                  return
22          # key handling
23          keys = pygame.key.get_pressed()
24          if keys[pygame.K_LEFT]:
25              board.cursor.move_left()
26              pygame.time.wait(100)
27          if keys[pygame.K_RIGHT]:
28              board.cursor.move_right()
29              pygame.time.wait(100)
30          if keys[pygame.K_DOWN]:
31              board.cursor.move_down()
32              pygame.time.wait(100)
33          # Update the display
34          pygame.display.update()
35
36          # Tick
37          clock.tick(60)
38
39  if __name__ == "__main__":
40      main()
```

矢印キーでカーソルを動かせるようになりました。pygame.time.wait(100) という見慣れない文があるかもしれませんが、これは 100 ミリ秒待つという意味です。mb.sleep と似ていますね。

5.4　まとめ

いろいろ考えた結果、今回はカーソルは Board の一部ということにして、カーソルの描画は Board に任せることにしました。さらに、キー入力を受け取るクラスを作ることも考えましたが、今回は main 関数で受け取ることにしました。このような設計は先を見据えて行うことが重要ですが、慣れないうちは「とりあえず動くもの」を作ることも大事です[*2]。

先生と考えよう

キー入力を受け取るクラスを作る場合、どのような設計になるでしょうか？

[*2] 今回の設計も正しいとは言い切れません。Board クラスが「Blob(ブロブ)」状態に陥る兆しがあります。詳しく知りたい人は調べたり聞いてみたりしてください。