

テトリス on Pygame 後編

Swimmy 高田馬場校

2024 年 8 月 30 日

目次

第 6 章	テトリスのブロックを作る	5
6.1	ブロックの設計	5
6.2	ブロックの表示方法を考える	5
6.3	ブロックのクラスを定義する	9
6.4	ブロックを表示させてみる	11
6.5	まとめ	20
第 7 章	回転するブロックを作る – T ブロック	21
7.1	T ブロック	21
7.2	T ブロックを動かす	29
7.3	回転できるか判定する関数をつくる	32
7.4	まとめ	37
第 8 章	他のブロックを作る	39
8.1	LBlock クラス	39
8.2	JBlock クラス	40
8.3	SBlock クラス	41
8.4	ZBlock クラス	42
8.5	IBlock クラス	43
第 9 章	ブロックの落下とランダムなブロックの生成	45
9.1	ブロックの落下	45
9.2	ランダムなブロックの生成	48
第 10 章	次のブロックへ切り替える	51
10.1	ブロックが一番下まで落ちた時とは	51
10.2	ブロックを積む	53

第 6 章

テトリスのブロックを作る

この章からはプログラムの例を示しません。今までのファイルに追記していく形になります。

6.1 ブロックの設計

6.1.1 ブロックの機能を考える

ブロックは、テトリスの中心的な要素です。ブロックはどんな機能を持つといいでしょうか？

ブロックの機能

- ブロックを回転させる関数
- ブロックを指定した位置に描画する関数

6.2 ブロックの表示方法を考える

6.2.1 設計

今回の設計では描画する関数は `Board` に任せられています。しかし、`Board` はどんな形のブロックを書くのかわからないのでどうにかしてブロックの情報をやり取りする必要があります。よって、設計する際にはブロックのクラス、`Board` のクラス両方に機能を加えます。

設計の案

- Board が現在保持中のブロックの変数を持つ
- Board はその変数に情報を要求して、その変数は塗らなければいけないマスの座標をリストで返す。これを `block_info` 関数とする。
- Board はそのリストをもとに、`draw` の途中で画面に書き込むことにする

図にするとこんな感じです。

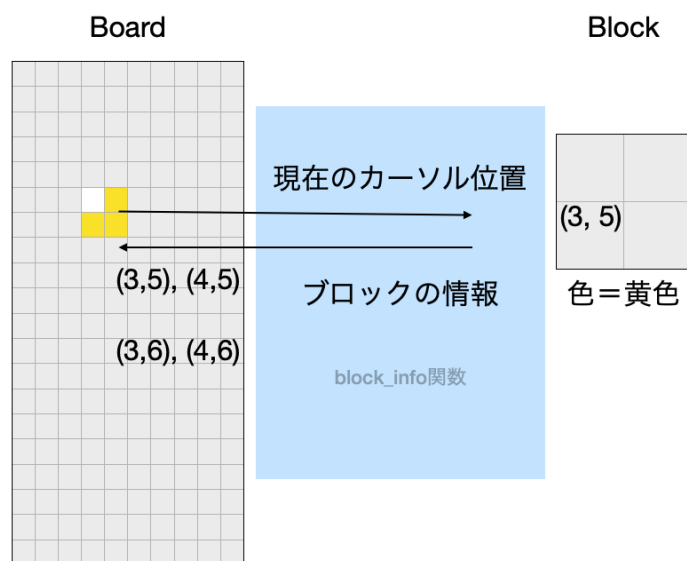


図 6.1 Board と Block のやりとり

Board が現在保持中のブロックの変数を持つということなので、Board クラスに変数を追加します。最初は何も持っていないので、「動いているブロック」という意味の「moving_block」に Python の特別な空を表す「None」を入れておきます。さらに、ブロックの描画を行う処理を Board クラスに追加します。画面に書く処理は `draw` 関数でしたね。draw 関数は現在の盤面を塗る（動かしているブロックは対象外）→カーソルを描く→グリッドを描く、という順番で行われていました。どこに動かしているブロックを描く処理を入れるか考えてみましょう。

ソースコード 6.1 Board クラスの変更点

```
1 class Board:
2     def __init__(self, tile_size):
3         # 作る時にタイルサイズを指定する
```

```
4         self.board = [[BLACK for _ in range(WIDTH)] for _ in range(
                    HEIGHT)]
5         self.TILE_SIZE = tile_size
6
7         # Cursor
8         self.cursor = Cursor()
9
10        # chapter6 ブロックを動かす
11        # 新たにmoving_block という変数を追加します
12        # 最初は何も動かしていないのでNoneを入れておきます
13        # わからない時は 3.3.1を見てみましょう
14        """この行を消して書いてみてください"""
15
16    def draw(self, screen):
17        for y in range(HEIGHT):
18            for x in range(WIDTH):
19                pygame.draw.rect(screen, self.board[y][x], (x * self.
                    TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE,
                    self.TILE_SIZE))
20
21        # Draw the cursor
22        for y in range(HEIGHT):
23            if self.board[y][self.cursor.x] == BLACK:
24                pygame.draw.rect(screen, CURSOR_COLOR, (self.cursor.x *
                    self.TILE_SIZE, y * self.TILE_SIZE, self.
                    TILE_SIZE, self.TILE_SIZE))
25
26        # chapter6 ブロックを動かす
27        # ブロックを動かす処理を追加します
28        # 動かすブロックがNoneではない時
29        if """何が入るでしょうか""":
30            # ブロックの情報を取得します
31            # 座標がリストになって帰ってくるはずなので変数に入れておきま
            す
32            """この行を消して書いてみてください"""
33
34            # 何色で塗るかはBlock の color という変数に入っています
35            # 一旦変数にコピーしておきます
36            """この行を消して書いてみてください"""
37
```

```

38         # ブロックからもらった座標リストを一つずつ見ていきます
39         # リストを一つずつ見ていくループはfor 文を使います
40         for """好きな変数の名前""" in """もらった座標リストを入れた
            変数""":
41             # for 文で決めた変数に座標が一つずつ入ってループされます
42             # それぞれの座標の場所を塗ることが目標です
43
44             # その 0 番目にx 座標が入っているのでそれを x に入れます
45             # 1 番目にy 座標が入っているのでそれを y に入れます
46             x = """変数の名前"""[0]
47             y = """変数の名前"""[1]
48
49             # マスを塗りつぶします。この行はわからなくて大丈夫ですが
50             # 色を入れた変数の名前を置き換えてください
51             pygame.draw.rect(screen, """34行目の色の変数の名前""",
                               (x * self.TILE_SIZE, y * self.TILE_SIZE, self.
                                TILE_SIZE, self.TILE_SIZE))
52
53         # あとは前回と同じです
54         # 順番を間違えるとブロックが線を上書きして表示されてしまうので
55         # この順番です。書き足す位置を間違えないように気をつけてください
56         # Draw the grid
57         for y in range(HEIGHT):
58             pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (
                WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
59         for x in range(WIDTH):
60             pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x
                * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))
61
62         def window_size(self):
63             return (WIDTH * TILE_SIZE, HEIGHT * TILE_SIZE)

```

"""..."""は自分で書いてみましょう。変数の名前を変にしすぎると先生が助けにくくなります。

6.3 ブロックのクラスを定義する

6.3.1 OBlock クラスを定義する

今回は O ブロックを作ります。O ブロックは一番簡単なブロックです。2x2 の黄色の正方形です。最初にこれから作ることにしたのは、回転が必要ないからです。ブロックにはどのような機能が必要だったか思い出してみます。

- ブロックを回転させる関数
- 自分の色をもつ変数
- ブロックの情報を返す関数

ソースコード 6.2 OBlock クラスの作成

```
1 class OBlock:
2     def __init__(self):
3         # 色は黄色です
4         # クラスの中にcolorという変数を作って黄色を入れておきます
5         """この行を消して書いてみてください"""
6
7     # Board からカーソルをもらってブロックの情報を計算して
8     # Board に描画するための座標リストを返します
9     def block_info(self, cursor):
10        # 最初はどうなるかわからないので [] を入れておきます
11        # 後から座標を入れて、最後に返すようにします
12        # 普通の変数result に [] を入れましょう
13        """この行を消して書いてみてください"""
14
15        # カーソルの座標を取得します
16        # カーソルの座標はcursor.x と cursor.y で取得できます
17        # それを変数に入れておきます
18        x = """上にした通り"""
19        y = """上にした通り"""
20
21        # 4つの座標を計算します
22        # まずは左上の座標です
23        # これはカーソルの座標そのままです
24        # 座標は (x 座標, y 座標) の形でリストに入れます
25        """10行目くらいに作った変数名""".append((x, y))
26
27        # 次に右上の座標です
28        # これは左上の座標から右に 1つ進んだ座標です
29        # x 座標を 1つ増やして y 座標はそのままです
30        # リストに入れます
31        """10行目くらいに作った変数名""".append("""ここを書いてみてくだ
32            さい""")
33
34        # 次に左下の座標です
35        # これは左上の座標から下に 1つ進んだ座標です
36        # x 座標はそのまま y 座標を 1つ増やします
37        # リストに入れます
38        """10行目くらいに作った変数名""".append("""ここを書いてみてくだ
39            さい""")
```

```
38
39     # 最後に右下の座標です
40     # これは左上の座標から右に 1つ進んで下に 1つ進んだ座標です
41     # x 座標を 1 つ増やして y 座標も 1 つ増やします
42     # リストに入れます
43     """10行目くらいに作った変数名""".append("""ここを書いてみてくだ
        さい""")
44
45     # 4つの座標が入ったリストを返します
46     return """10行目くらいに作った変数名"""
47
48     # クラスの中に回転する関数を作ってみます
49     # 今回は回転しないので何も書きませんが、作り方だけ復習
50     # rotate という関数を作ります
51     # 関数の中身は...としておきます
52     """3.4を参考にしてください"""
```

block_info 関数はカーソルの位置を受け取って、そこからカーソルの位置、右、下、右下の4つのマスを返します。Board はその座標と OBlock の中にある color 変数を使って描画します。rotate 関数は今回は作っていませんが、ブロックの種類によっては必要になります。

6.4 ブロックを表示させてみる

6.4.1 main 関数を変更する

main 関数を変更して、OBlock を動かしてみましょう。

ソースコード 6.3 OBlock のテスト

```
1 import pygame
2 from tetris import Board
3 # chapter 6 ブロックを動かす
4 # OBlock をインポートします
5 from tetris import OBlock
6
7 def main():
8     # Create the board
9     board = Board(tile_size=30)
10
11     # Initialize pygame
```

```
12     pygame.init()
13     screen = pygame.display.set_mode(board.window_size())
14     pygame.display.set_caption("Tetris")
15     clock = pygame.time.Clock()
16
17     # chapter 6 ブロックを動かす
18     # とりあえず0 ブロックを作っておきます
19     # 将来はランダムに作ります
20     board.moving_block = OBlock()
21
22     while True:
23         # Draw the board
24         board.draw(screen)
25
26         # Handle events
27         for event in pygame.event.get():
28             if event.type == pygame.QUIT:
29                 return
30
31         # key handling
32         keys = pygame.key.get_pressed()
33         if keys[pygame.K_LEFT]:
34             board.cursor.move_left()
35             pygame.time.wait(100)
36         if keys[pygame.K_RIGHT]:
37             board.cursor.move_right()
38             pygame.time.wait(100)
39         if keys[pygame.K_DOWN]:
40             board.cursor.move_down()
41             pygame.time.wait(100)
42         # Update the display
43         pygame.display.update()
44
45         # Tick
46         clock.tick(60)
47
48     if __name__ == "__main__":
49         main()
```

うまくいっていれば、O ブロックが動くはずです。

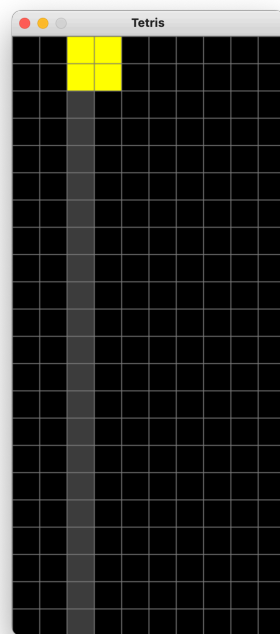


図 6.2 テトリスの実行結果

ここで右端に動いてみましょう。ブロックが半分消えてしまいますね。

6.4.2 ブロックの移動範囲を制限する

この問題を解決するために、ブロックの移動範囲を制限する機能を追加します。キー入力を受け取ってから動くまでの間に、ブロックが動けるかどうかを判定する処理を入れます。さて、どこに入れるといいでしょうか？

設計

- main 関数が判定する
- Board が判定する
- Cursor が判定する
- OBlock が判定する

クラスを使う前までは main 関数が判定するのが一般的でした。しかし、今回は main には余計な仕事をさせないようにしましょう。動きを担当するのは Cursor の役割です。Cursor に move_right 関数などを追加していたはずですが、ここを次のように変更することで対処します。

- OBlock は Cursor, Board から動けるかどうか計算する
- Cursor はその情報をもとに動くかどうか決める

6.4.3 OBlock クラスを変更する

OBlock クラスに、動ける範囲を計算する関数を追加します。

ソースコード 6.4 OBlock クラスの変更

```
1 class OBlock:
2     def __init__(self):
3         self.color = YELLOW
4
5     def block_info(self, cursor):
6         """省略"""
7
8     def rotate(self):
9         pass
10
11     # 左に動けるかを返すcan_go_left 関数
```

```

12     # 現在の座標を知るために
        cursor を、盤面の情報を知るために board_data を引数に取ります
13     def can_go_left(self, cursor, board_data):
14         # 今回は0 ブロックなので、cursor が 0 (左端) だと左に動けません
15         if cursor.x == 0:
16             return False
17         # あと、カーソルの左と左下にブロックがあるときも左に動けません
18         # ブロックが入っているということは、何かの色が入っているということ
            です
19         if board_data[cursor.y][cursor.x - 1] != BLACK:
20             return False
21         if board_data[cursor.y + 1][cursor.x - 1] != BLACK:
22             return False
23         # ここまで来たということは左に動けるということです
24         return True
25
26     # 右に動けるかを返す can_go_right 関数
27     # 同じように作ってみましょう
28     """この行を消して書いてみてください"""

```

さて、次にカーソルの動きを変更します。今までは無条件に動いていましたが、一度ブロックを受け取って動けるかどうかを判定してもらい、動けるなら動くようにします。

6.4.4 Cursor クラスを変更する

ソースコード 6.5 Cursor クラスの変更

```

1 class Cursor:
2     def __init__(self):
3         self.x = WIDTH // 2
4         self.y = 0
5
6     # カーソルは動く時にblock という引数と
7     # board をもらうことにします
8     # こうしないとblock の
9     # can_go_left 関数が使えません
10    def move_left(self, block, board):
11        # もし、block が左に動けるなら
12        if """block が左に動けるかを返す関数""":
13            # カーソルを左に動かします

```

```

14         self.x = max(0, self.x - 1)
15
16     # 残りも変えてみましょう
17     def move_right(self):
18         self.x = min(WIDTH - 1, self.x + 1)
19     def move_down(self):
20         self.y = min(HEIGHT - 1, self.y + 1)
21     def move_up(self):
22         self.y = max(0, self.y - 1)

```

これらの変更が終わると、main.py にエラーが出ているはずです。今までは `cursor.move_right()` のように書いていましたが、これからは引数として `board` と `block` を渡す必要があるからです。もちろん、main 関数で引数を渡すように変更しても良いのですが、少し見栄えが悪いので（プログラマとしては見栄えが悪いです）、変更を加えます。

6.4.5 Board クラスを変更する

今回はカプセル化という方法で、Board クラスが Cursor をカプセルのように閉じ込めて、Board 経由で Cursor を操作するようにします。そこで、Board クラスに `move_right` 関数を追加し、Cursor の `move_right` 関数を呼び出すようにします。

ソースコード 6.6 Board クラスの変更

```

1 class Board:
2     def __init__(self, tile_size):
3         # 作る時にタイルサイズを指定する
4         self.board = [[BLACK for _ in range(WIDTH)] for _ in range(
5             HEIGHT)]
6         self.TILE_SIZE = tile_size
7
8         # Cursor
9         self.cursor = Cursor()
10
11        # chapter6 ブロックを動かす
12        # 新たにmoving_blockという変数を追加します
13        # 最初は何も動かしていないのでNoneを入れておきます
14        # わからない時は3.3.1を見てみましょう
15        """この行を消して書いてみてください"""
16
17    def draw(self, screen):
18        for y in range(HEIGHT):

```



```
18         for x in range(WIDTH):
19             pygame.draw.rect(screen, self.board[y][x], (x * self.
                TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE,
                self.TILE_SIZE))
20
21     # Draw the cursor
22     for y in range(HEIGHT):
23         if self.board[y][self.cursor.x] == BLACK:
24             pygame.draw.rect(screen, CURSOR_COLOR, (self.cursor.x *
                self.TILE_SIZE, y * self.TILE_SIZE, self.
                TILE_SIZE, self.TILE_SIZE))
25
26     # chapter6 ブロックを動かす
27     # ブロックを動かす処理を追加します
28     # 動かすブロックがNone ではない時
29     if """何が入るでしょうか""":
30         # ブロックの情報を取得します
31         # 座標がリストになって帰ってくるはずなので変数に入れておきま
            す
32         """この行を消して書いてみてください"""
33
34         # 何色で塗るかはBlock の color という変数に入っています
35         # 一旦変数にコピーしておきます
36         """この行を消して書いてみてください"""
37
38         # ブロックからもらった座標リストを一つずつ見ていきます
39         # リストを一つずつ見ていくループはfor 文を使います
40         for """好きな変数の名前""" in """もらった座標リストを入れた
            変数""":
41             # for 文で決めた変数に座標が一つずつ入ってループされます
42             # それぞれの座標の場所を塗ることが目標です
43
44             # その 0 番目にx 座標が入っているのでそれを x に入れます
45             # 1 番目にy 座標が入っているのでそれを y に入れます
46             x = """変数の名前"""[0]
47             y = """変数の名前"""[1]
48
49             # マスを塗りつぶします。この行はわからなくて大丈夫ですが
50             # 色を入れた変数の名前を置き換えてください
51             pygame.draw.rect(screen, """34行目の色の変数の名前""",
```

```

        (x * self.TILE_SIZE, y * self.TILE_SIZE, self.
         TILE_SIZE, self.TILE_SIZE))

52
53     # あとは前回と同じです
54     # 順番を間違えるとブロックが線を上書きして表示されてしまうので
55     # この順番です。書き足す位置を間違えないように気をつけてください
56     # Draw the grid
57     for y in range(HEIGHT):
58         pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (
59             WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
60     for x in range(WIDTH):
61         pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x
62             * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))
63
64
65     # chapter6 ブロックを動かす
66     # cursor の動く機能をカプセル化します
67     # cursor_move_left という関数を追加します
68     def cursor_move_left(self):
69         # cursor の move_left 関数を呼び出します
70         # 引数にはmoving_block と board を渡します
71         self.cursor.move_left(self.moving_block, self.board)
72
73     # 残りの動きも同じようにします

```

6.4.6 main 関数の変更

最後に、main 関数を変更して、cursor を使うのではなく、board の移動機能を使うようにします。

ソースコード 6.7 main 関数の変更

```

1 import pygame
2 from tetris import Board
3 # chapter 6 ブロックを動かす
4 # OBlock をインポートします
5 from tetris import OBlock
6

```

```
7 def main():
8     # Create the board
9     board = Board(tile_size=30)
10
11     # Initialize pygame
12     pygame.init()
13     screen = pygame.display.set_mode(board.window_size())
14     pygame.display.set_caption("Tetris")
15     clock = pygame.time.Clock()
16
17     # chapter 6 ブロックを動かす
18     # とりあえず0ブロックを作っておきます
19     # 将来はランダムに作ります
20     board.moving_block = OBlock()
21
22     while True:
23         # Draw the board
24         board.draw(screen)
25
26         # Handle events
27         for event in pygame.event.get():
28             if event.type == pygame.QUIT:
29                 return
30
31         # key handling
32         keys = pygame.key.get_pressed()
33         if keys[pygame.K_LEFT]:
34             # cursor の move_left メソッドをやめて
35             #
36             board の cursor_move_left メソッドを呼び出すように変更します
37
38             """board.cursor.move_left() -> ????"
39             pygame.time.wait(100)
40         if keys[pygame.K_RIGHT]:
41             # ここのように変更します
42             board.cursor.move_right()
43             pygame.time.wait(100)
44         if keys[pygame.K_DOWN]:
45             # ここのように変更します
46             board.cursor.move_down()
47             pygame.time.wait(100)
```

```
45         # Update the display
46         pygame.display.update()
47
48         # Tick
49         clock.tick(60)
50
51 if __name__ == "__main__":
52     main()
```

これを実行すると、ブロックが右端に行かなくなります。また、下方向にも消えずに止まるようになります。

6.5 まとめ

今回は、簡単な OBlock を作成しました。Board クラスにブロックの情報を持たせ、Board は適宜 OBlock に情報を要求して描画するようにしました。また、ブロックの移動範囲を制限する機能を追加し、Cursor クラスにその機能を持たせました。最後に、main 関数を変更して、Cursor ではなく Board を使ってブロックを動かすようにしました。

懺悔

正直にいうとこのテトロリスの設計、若干失敗したなあと思っています。今は Board が moving_block を持っています。しかし今回の章でいえば、ブロックを Cursor の中に持たせるべきでした。そうであれば Board が Cursor をカプセル化する必要もなかった上に設計が簡単になります。今後ブロックを落とす処理を追加するときを見越してこのような設計にしたのでいいのですが、果たしてこれが正解だったのか、今後の展開にご期待ください。

第 7 章

回転するブロックを作る – T ブロック

7.1 T ブロック

今回は T ブロックを作ります。T ブロックは、T の字の形をしています。T ブロックは、回転が必要なブロックです。回転するためには、ブロックの形を表す変数が必要です。今回は、ブロックの形を方角と対応させて持つことにしますが、**T ブロックの上の横棒が向いている方が方角になる**ようにします。また、カーソルの位置が T のくっついているところになるようにします。

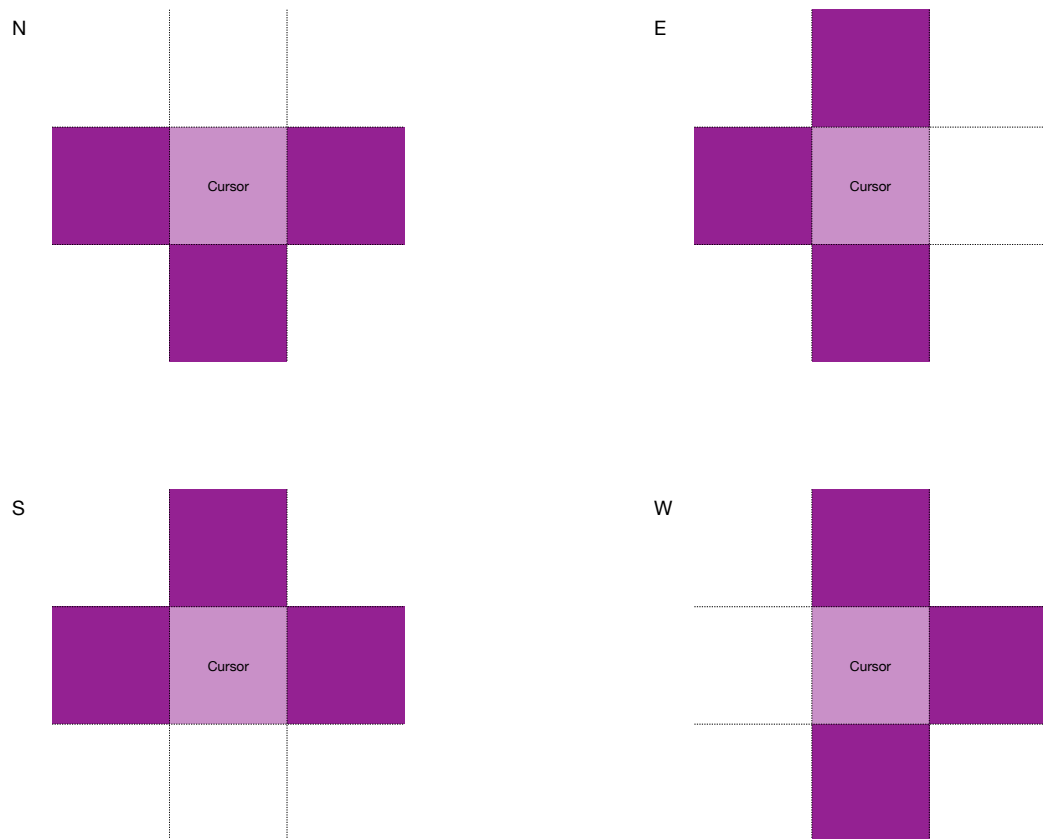


図 7.1 Tブロック

7.1.1 Tブロックのプログラム

ソースコード 7.1 Tブロックのプログラム

```

1 # もう作っていたら飛ばして大丈夫です
2 N = 0
3 E = 1
4 S = 2
5 W = 3
6
7 # OBlock の下あたりに書いてみましょう
8 # TBlock のクラスを作ります
9 # もうクラスの作り方は覚えましてしょうか
10 """この行を消して書いてみてください"""
11

```

```
12     # クラスが作られた時に呼ばれる関数を作ります
13     # 3.3.1を見ながら作ってみましょう
14     """クラスが作られた時に呼ばれる関数"""
15         # 色を決めます
16         # クラス内にcolor という変数を作って
17         # 色を入れておきます
18         # 色はネットで調べて入れてみましょう
19         """この行を消して書いてみてください"""
20
21         # 最初の方角を設定します
22         # クラスの中にrotation という変数を作って、
23         # 最初は北向きの変数を入れます
24         # ここで言う 2行目で作ったやつです
25         """方角を設定する行"""
26
27     # ブロックの情報を計算して返す関数を作ります
28     # OBlock と同様、表示されている位置が
29     # 必要なので、cursor を引数に入れます
30     def """ブロックの情報を計算して返す関数"""(self, cursor):
31         # 最初はどうかかわからないので [] を入れておきます
32         # 後から座標を入れて、最後に返すようにします
33         # 普通の変数result に[]を入れましょう
34         """この行を消して書いてみてください"""
35
36         # カーソルの座標を取得します
37         # カーソルの座標はcursor.x と cursor.y で取得できます
38         # それを変数に入れておきます
39         x = """上に書いた通り"""
40         y = """上に書いた通り"""
41
42         # ここからはTBlock の座標計算です
43         # 自分の向きを見ます
44         # もし北向きだったら
45         if self.rotation == N:
46             # 4つの座標を計算します
47             # まずは中心の座標です
48             # これはカーソルの座標そのままです
49             # 座標は (x 座標, y 座標) の形でリストに入れます
50             """34行目くらいに作った変数名""".append((x, y))
51
```

```
52         # 次に上の座標です
53         # これは中心の座標から上に 1つ進んだ座標です
54         """34行目くらいに作った変数名""".append((x, y - 1))
55
56         # 次に左の座標です
57         # これは中心の座標から左に 1つ進んだ座標です
58         """34行目くらいに作った変数名""".append((x - 1, y))
59
60         # 次に右の座標です
61         # これは中心の座標から右に 1つ進んだ座標です
62         """34行目くらいに作った変数名""".append((x + 1, y))
63     # もし東向きだったら
64     elif """ここを書いてみてください""":
65         # 4つの座標を計算します
66         # まずは中心の座標です
67         # これはカーソルの座標そのままです
68         # 座標は (x 座標, y 座標) の形でリストに入れます
69         """34行目くらいに作った変数名""".append((x, y))
70
71         # 次に上の座標です
72         # これは中心の座標から上に 1つ進んだ座標です
73         """34行目くらいに作った変数名""".append("""ここを書いてみて
74             ください""")
75
76         # 次に下の座標です
77         # これは中心の座標から下に 1つ進んだ座標です
78         """34行目くらいに作った変数名""".append("""ここを書いてみて
79             ください""")
80
81         # 次に左の座標です
82         # これは中心の座標から左に 1つ進んだ座標です
83         """34行目くらいに作った変数名""".append("""ここを書いてみて
84             ください""")
85     # もし南向きだったら
86     elif """ここを書いてみてください""":
87         # 4つの座標を計算します
88         # まずは中心の座標です
89         # これはカーソルの座標そのままです
90         # 座標は (x 座標, y 座標) の形でリストに入れます
91         """34行目くらいに作った変数名""".append((x, y))
```



```
90         # 次に上の座標です
91         # これは中心の座標から上に 1つ進んだ座標です
92         """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
93
94         # 次に左の座標です
95         # これは中心の座標から左に 1つ進んだ座標です
96         """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
97
98         # 次に右の座標です
99         # これは中心の座標から右に 1つ進んだ座標です
100        """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
101
102        # もし西向きだったら
103        elif """ここを書いてみてください""":
104            # 4つの座標を計算します
105            # まずは中心の座標です
106            # これはカーソルの座標そのままです
107            # 座標は (x 座標, y 座標)の形でリストに入れます
108            """34行目くらいに作った変数名""".append((x, y))
109
110            # 次に上の座標です
111            # これは中心の座標から上に 1つ進んだ座標です
112            """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
113
114            # 次に下の座標です
115            # これは中心の座標から下に 1つ進んだ座標です
116            """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
117
118            # 次に右の座標です
119            # これは中心の座標から右に 1つ進んだ座標です
120            """34行目くらいに作った変数名""".append("""ここを書いてみて
           ください""")
121
122        # 方角がなんであっても
123        # 34行目くらいに作った変数には 4つの座標が入っています
124        # それを返します
125        return """34行目くらいに作った変数名"""
```

```
126
127     # クラスの中に回転する関数を作ってみます
128     # 関数名はrotate とします
129     def """回転する関数"""(self):
130         # 方角を変えます
131         # 方角はN, E, S, W の順番で変えていきます
132         # 一番最後にW になったら N に戻します
133         # 描画も更新しないといけないような気がしますが、
134         # 方角さえ変えてしまえば次にblock_info が呼ばれる時に
135         # 描画が変わるので大丈夫です
136         """方角を変える行"""
```

プログラミング豆知識

回転系、周期系の変数の処理を楽に書きたいときは、それらを 0 から始まる数字の連番にすることと、あまりが繰り返すことを利用すると上手く書けます。

ソースコード 7.2 方角を扱う

```
1 N=0
2 E=1
3 S=2
4 W=3
5 direction = N
6 direction = (direction + 1) % 4 # direction は E(1)になる
7 direction = (direction + 1) % 4 # direction は S(2)になる
8 direction = (direction + 1) % 4 # direction は W(3)になる
9 direction = (direction + 1) % 4 # direction は N(0)になる、3+1は 4だが
    、4を 4で割った余りは 0なため
```

このように、N から始まった方角がまた N に戻ってきます。

7.1.2 T ブロックの描画

T ブロックの描画は、O ブロックと同じように行います。O ブロックは Board に block_info を求められ、その情報をもとに描画していました。でも、T ブロックも block_info を持っていますから、O ブロックと同じように描画できます。つまり、**Board クラスは変更が必要ありません。オブジェクト指向の利点です**。違うクラスであっても同じ名前で関数を設計すると、他のクラスから同じように扱えます。でも中身自体は違うので、それぞれのブロックがそれぞれにあった情報を返すことができるのです。こういう性質を「ポリモーフィズム/多態性」と言います*¹。では、main 関数を変更して最初に T ブロックを表示してみましょう。

ソースコード 7.3 テスト用に T ブロックを表示

```
1 import pygame
2 from tetris import Board
3 # chapter 7 TBlock
4 # TBlock をインポートします
5 from tetris import OBlock, TBlock
```

*¹ 厳密には違うんですが Python にポリモーフィズムはあってないようなものなので教えるとしたらこうするしかないんです

```
6
7 def main():
8     # Create the board
9     board = Board(tile_size=30)
10
11    # Initialize pygame
12    pygame.init()
13    screen = pygame.display.set_mode(board.window_size())
14    pygame.display.set_caption("Tetris")
15    clock = pygame.time.Clock()
16
17    # chapter 7 T ブロック
18    # 試しにT ブロックに変更してみましょう
19    board.moving_block = """クラスを変数につくる"""()
20
21    while True:
22        # Draw the board
23        board.draw(screen)
24
25        # Handle events
26        for event in pygame.event.get():
27            if event.type == pygame.QUIT:
28                return
29        # key handling
30        keys = pygame.key.get_pressed()
31        if keys[pygame.K_LEFT]:
32            board.cursor_move_left()
33            pygame.time.wait(100)
34        if keys[pygame.K_RIGHT]:
35            board.cursor_move_right()
36            pygame.time.wait(100)
37        if keys[pygame.K_DOWN]:
38            board.cursor_move_down()
39            pygame.time.wait(100)
40        # Update the display
41        pygame.display.update()
42
43        # Tick
44        clock.tick(60)
45
```

```
46 if __name__ == "__main__":  
47     main()
```

できたら実行しましょう。表示だけで動かすとエラーになります。

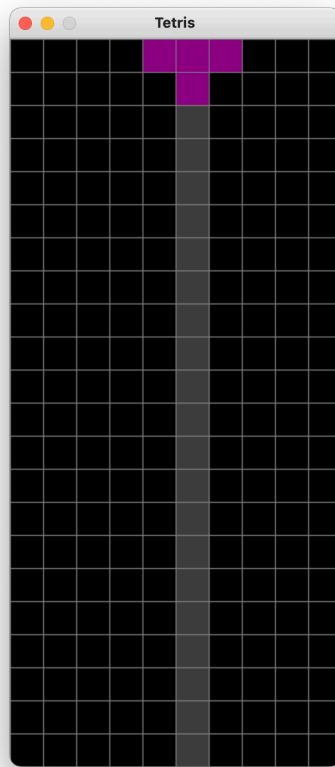


図 7.2 T ブロックの表示

7.2 T ブロックを動かす

なぜエラーになったのでしょうか？それは、T ブロックが動けるかどうかを判定する機能がないからです。O ブロックは動けるかどうかを判定する機能を持っていましたが、T ブロックにはありません。Board はそのことを知らずにその関数を呼び出してしまったためエラーになります。T ブロックにも動けるかどうかを判定する機能を追加しましょう。

7.2.1 T ブロックに動けるかどうかを判定する機能を追加する

T ブロックにも O ブロックと同じように、動けるかどうかを判定する機能を追加します。

ソースコード 7.4 Tブロックに動けるかどうかを判定する機能を追加

```

1  # もう作っていたら飛ばして大丈夫です
2  N = 0
3  E = 1
4  S = 2
5  W = 3
6
7  # OBlock の下あたりに書いてみましょう
8  # TBlock のクラスを作ります
9  # もうクラスの作り方は覚えましてでしょうか
10 """この行を消して書いてみてください"""
11
12     """ 省略 rotate 関数の下ぐらいに書いてください """
13
14     # OBlock と名前を同じにしないといけないので気をつけてください。
15     # 左に動けるか判定する関数を作ります
16     def """左に動けるか判定する関数"""(self, cursor, board_data):
17         if """方角が N の時""" :
18             # 左に 1マスは必要なので、左端 1マス前には動けません
19             if cursor.x == 1:
20                 return False
21
22             # 左に 1マス動いた時に壁に当たるか判定します
23             # 本体ブロックは左に一つあるので、左に 2つ目が壁かどうかを見ま
24             # す
25             if board_data[cursor.y][cursor.x - 2] != BLACK:
26                 return False
27             # 下のマスも壁に当たるか判定します
28             if board_data[cursor.y + 1][cursor.x - 1] != BLACK:
29                 return False
30             return True
31         elif """方角が E の時""" :
32             # ここも同じように考えてください
33             # 方角がE ということは-|の形になっています
34             """この行を消して書いてみてください"""
35             """方角が S の時と W の時も書いてみてください"""
36         def """右に動けるか判定する関数"""(self, cursor, board_data):
37             """左に動けるか判定する関数を参考に書いてみましょう"""
38
39         def """下に動けるか判定する関数"""(self, cursor, board_data):

```

```
39         """左に動けるか判定する関数を参考に見てみましょう"""
40
41     def """上に動けるか判定する関数"""(self, cursor, board_data):
42         """左に動けるか判定する関数を参考に見てみましょう"""
```

7.2.2 あれ、回転は？

main 関数でキー入力を受け取っていますので、main 関数を実行して回転キーを設定しましょう。

ソースコード 7.5 main 関数で回転する

```
1 def main():
2     board = Board(tile_size=30)
3
4     pygame.init()
5     screen = pygame.display.set_mode(board.window_size())
6     pygame.display.set_caption("Tetris")
7     clock = pygame.time.Clock()
8
9     board.moving_block = TBlock()
10
11     while True:
12         # Draw the board
13         board.draw(screen)
14
15         # Handle events
16         for event in pygame.event.get():
17             if event.type == pygame.QUIT:
18                 return
19         # key handling
20         keys = pygame.key.get_pressed()
21         if keys[pygame.K_LEFT]:
22             board.cursor_move_left()
23             pygame.time.wait(100)
24         if keys[pygame.K_RIGHT]:
25             board.cursor_move_right()
26             pygame.time.wait(100)
27         if keys[pygame.K_DOWN]:
28             board.cursor_move_down()
```

```
29         pygame.time.wait(100)
30
31     # chapter 7 T ブロック
32     # T ブロックを回転させる
33     if keys[pygame.K_r]:
34         # board 中にある moving_block の rotate 関数を呼び出します
35         # 今回は書いておきました
36         board.moving_block.rotate()
37         pygame.time.wait(100)
38     # Update the display
39     pygame.display.update()
40
41     # Tick
42     clock.tick(60)
43
44 if __name__ == "__main__":
45     main()
```

書き終えたら実行してみましょう。ひとまずお疲れ様でした。でも、まだ終わりではありません。

回転によりはみ出してしまうことがありますね。これは移動と同じで回転できない状況があるのに、それを検知できずに回転しているからです。

Q: この教材、いきあたりばったりで作ってないですか？

多くのメンターさんが共感してくれると思いますが、基本的にプログラムは一発で動きません。さらに残酷なことに、頑張って書いた時ほど、動かないものです。ここまできたみなさんには、同じプログラマとして、「せっかく頑張ったのになんでやねん」というこのガッカリ感を味わってみて欲しいです。

実行する前にこうなることが予期できた人は素晴らしいです。ぜひ、次回以降もこのような予測をしていってください。

7.3 回転できるか判定する関数をつくる

それでは、回転について判定する関数を作ってみましょう。いくつか方法が考えられますが、今回は、回転した後の座標を計算して、その座標が盤面と衝突しないかどうかを判定する方法を取ります。

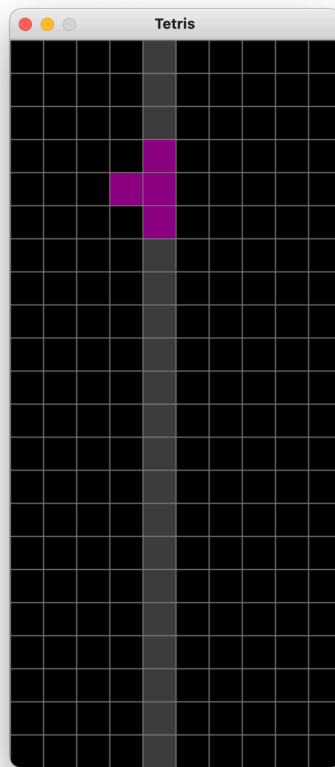


図 7.3 T ブロックの回転

7.3.1 回転できるか判定する関数をつくる

以下のように書いてみましょう。戻り値は bool 型、すなわち True か False です。可能だと分かったらその時点で return True、逆に不可能だと分かったら return False します。

ソースコード 7.6 回転できるか判定する関数をつくる

```
1 class TBlock:
2     """省略、一番下に書いてください"""
3     # 回転できるか判定する関数です
4     # カーソルの位置によって回転できるかは
5     # 変わってくるので、引数にカーソルを取ります
6     # また、board_data で盤面の状態を取得します
7     def can_rotate(self, cursor, board_data):
8         if self.rotation == N:
9             # 北向きから東向きに回転する場合
10            # T -> -| のようになります
```

```

11         """回転によりカーソルが盤面外に出るか判定します"""
12         """4つのマスについて調べて、回転できるか判定します"""
13
14         elif self.rotation == E:
15             # 東向きから南向きに回転する場合
16             # T -> | のようになります
17             """同様に行います"""
18             """4つのマスについて調べて、回転できるか判定します"""
19         elif ...:
20             """同様に行います"""

```

ちなみに、もっと賢い方法を思いついた人は、それを書いて試してみましょう。この教材では直感的な方法をとっています。

次に、判定をしてから回転する部分を作ります。今回も前回同様に、Board クラスがブロックの回転する関数を提供します。

7.3.2 Board クラスに回転する関数をつくる

ソースコード 7.7 Board クラスに回転する関数をつくる

```

1 class Board:
2     """省略"""
3     """一番下あたりに書いてください"""
4     def block_rotate(self):
5         if self.moving_block.can_rotate(self.cursor, self.board_data):
6             self.moving_block.rotate()

```

これで回転できる時に回転する、機能が完成しました。最後に、main 関数を変更しましょう。

7.3.3 main 関数を変更する

ソースコード 7.8 main 関数を変更する

```

1 import pygame
2 from tetris import Board
3 from tetris import OBlock, TBlock
4
5 def main():
6     # Create the board

```

```
7     board = Board(tile_size=30)
8
9     # Initialize pygame
10    pygame.init()
11    screen = pygame.display.set_mode(board.window_size())
12    pygame.display.set_caption("Tetris")
13    clock = pygame.time.Clock()
14
15    board.moving_block = TBlock()
16
17    while True:
18        # Draw the board
19        board.draw(screen)
20
21        # Handle events
22        for event in pygame.event.get():
23            if event.type == pygame.QUIT:
24                return
25        # key handling
26        keys = pygame.key.get_pressed()
27        if keys[pygame.K_LEFT]:
28            board.cursor_move_left()
29            pygame.time.wait(100)
30        if keys[pygame.K_RIGHT]:
31            board.cursor_move_right()
32            pygame.time.wait(100)
33        if keys[pygame.K_DOWN]:
34            board.cursor_move_down()
35            pygame.time.wait(100)
36        if keys[pygame.K_r]:
37            # ch7 ブロックが回転できるか判定する
38            """変更前"""
39            board.moving_block.rotate()
40            """変更後"""
41            board.block_rotate()
42            pygame.time.wait(100)
43        # Update the display
44        pygame.display.update()
45
46        # Tick
```

```

47         clock.tick(60)
48
49 if __name__ == "__main__":
50     main()

```

実行すると、T ブロックが正しく回転するようになるはずです。うまくいかない場合は `can_rotate` が大体間違っていると思うので、先生と相談してください。

コラム: if __name__ == "__main__" について

Swimmy の教材には書いていませんが、Python には `__name__` という変数があらかじめ使えます。もちろん変数なので `print` が可能です。

ソースコード 7.9 __name__ の使い方

```

1 print(__name__)

```

これを実行すると、`__main__` と表示されます。これは、このファイルが直接実行されたときに `__main__` になるということです。Python ファイルには、実行する方法が2度あります。

- 直接実行する
- `import` して使う

`import` された時には、`__name__` はファイル名になります。`import pygame` をしたときに

```
pygame 2.0.1 (SDL 2.0.14, Python 3.8.3)
```

```
Hello from the pygame community. https://www.pygame.org/contribute.html
```

こんな表示を見たことはありませんか？おそらく、`pygame` のファイルにはこんなのを書いてあるはずです

ソースコード 7.10 pygame のファイルの一部

```

1 print("pygame_2.0.1_(SDL_2.0.14,_Python_3.8.3)")
2 print("Hello_from_the_pygame_community._https://www.pygame.org/
    contribute.html")

```

使い道はかなり限定されていますが、

- このファイルが間違って `import` された時に警告を出したい
- `import` して使って欲しいので直接実行された時はエラーを出したい
- バージョンを表示したり、著作権表示をしたい

こんな時に使われている傾向があります。せっかくなので、自分のファイルにも書いてみてください。

```
1 # 今回はtetris.pyの中身を変更します。
2 # 一番下あたりに次のようなコードを追加します。
3 if __name__ == "__main__":
4     print("このプログラムは単体で実行できません。")
5 else:
6     print("このプログラムは「自分の名前」が作りました。")
```

import tetris のあたりで print が実行され、自分の名前が出てくるはずです。

7.4 まとめ

今回は T ブロックを作りました。T ブロックは O ブロックと違い、回転が必要なブロックです。そのため、回転できるかどうかを判定する機能を追加しました。

第 8 章

他のブロックを作る

8.1 LBlock クラス

今までの T ブロックを元に、L ブロックを作ります。クラス名は LBlock としましょう。

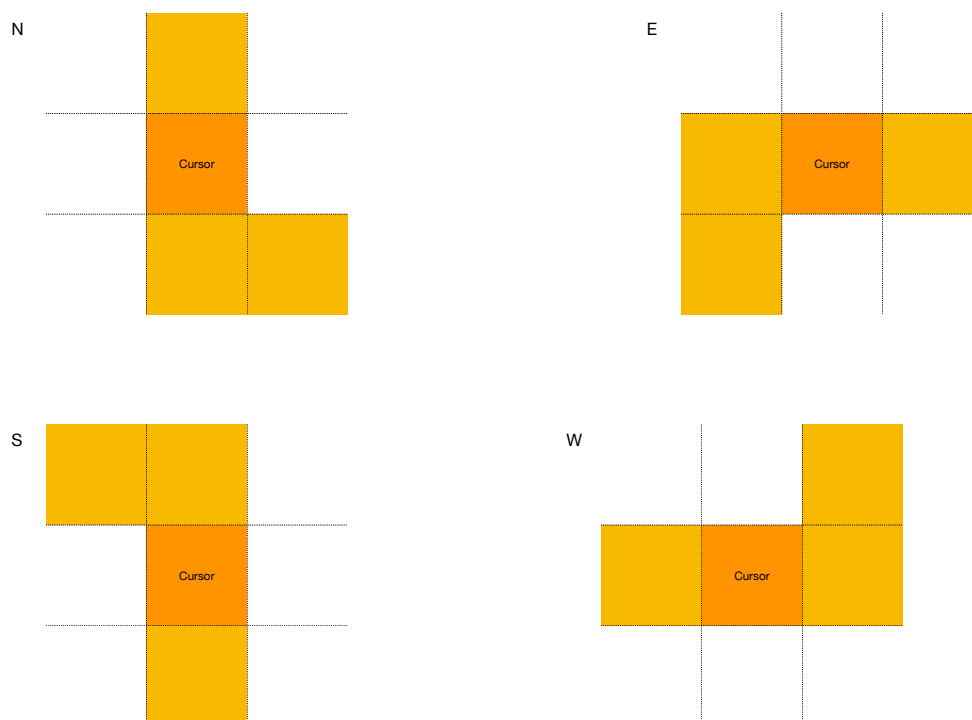


図 8.1 L ブロック

色はオレンジが多いようです。以下の関数を作るのを忘れないでください。

- block_info 関数 ... カーソルの位置から自分のブロックの様子を座標のリストで返

します。

- `rotate` 関数 ... 回転します。回転できるかは気にせず、とりあえず `self` に入っている `rotation` という変数を変えるだけにします。
- `can_rotate` 関数 ... 実際に回転したときにはみ出さないか、他のブロックとぶつからないか判定します。引数 `cursor` で現在の位置を、引数 `board_info` で盤面の情報を取得します。
- `can_go_up` 関数 ... 上に移動できるか判定します。引数 `cursor` で現在の位置を、引数 `board_info` で盤面の情報を取得します。
- `can_go_down` 関数 ... 下に移動できるか判定します。引数 `cursor` で現在の位置を、引数 `board_info` で盤面の情報を取得します。
- `can_go_right` 関数 ... 右に移動できるか判定します。引数 `cursor` で現在の位置を、引数 `board_info` で盤面の情報を取得します。
- `can_go_left` 関数 ... 左に移動できるか判定します。引数 `cursor` で現在の位置を、引数 `board_info` で盤面の情報を取得します。

出来上がったら、`main` 関数で表示してみましょう。`main.py` の `moving_block =` の部分を変更したらできるはずです。

8.2 JBlock クラス

次にJブロックを作ります。関数も同じものを作ります。自分を信じすぎず、一つブロックを作ったらきちんと表示、回転、移動ができるか確認しましょう。

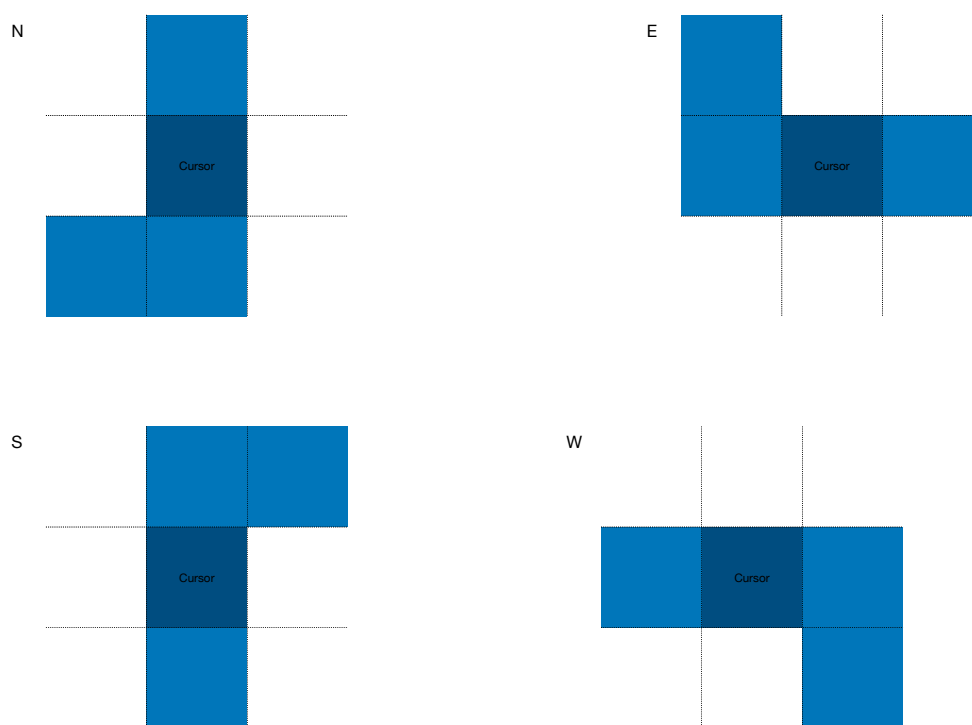


図 8.2 J ブロック

8.3 SBlock クラス

次に S ブロックを作ります。関数も同じものを作ります。

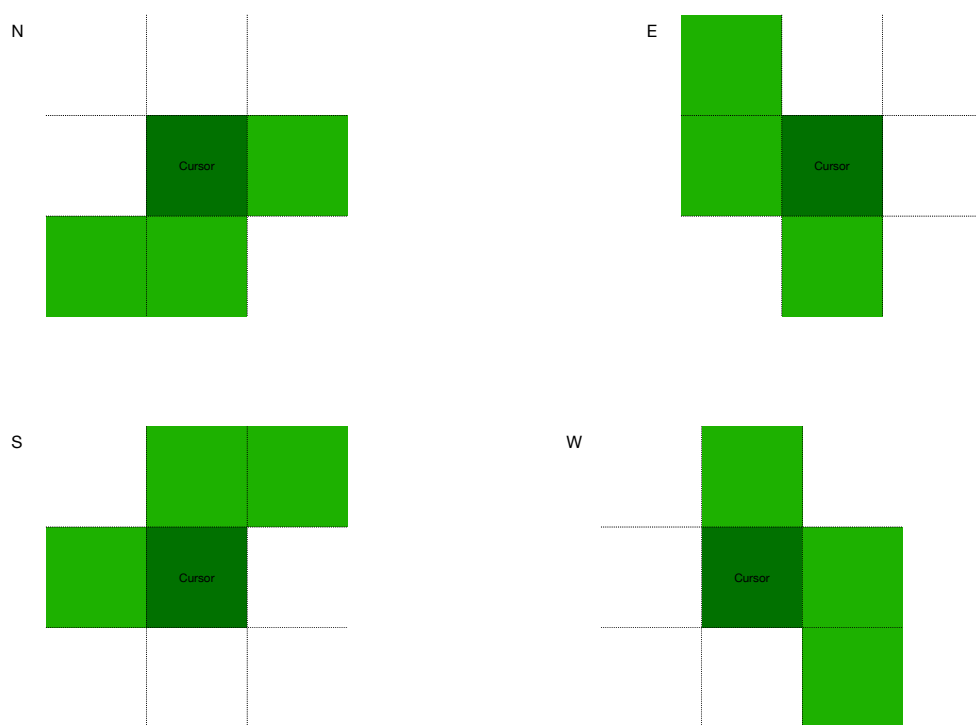


図 8.3 S ブロック

8.4 ZBlock クラス

次に Z ブロックを作ります。

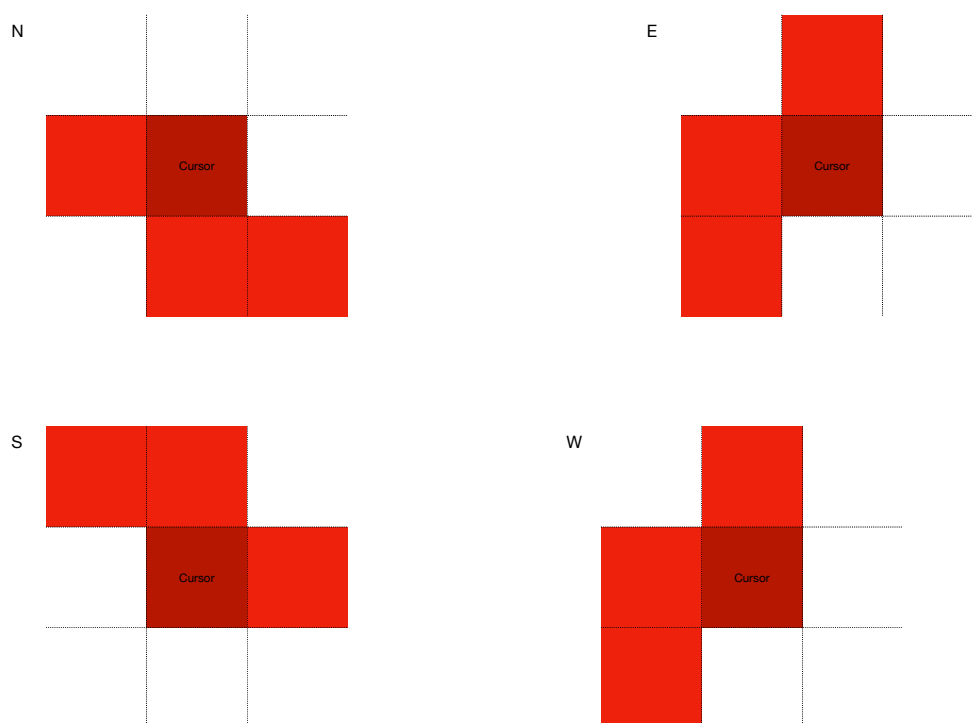


図 8.4 Z ブロック

8.5 IBlock クラス

最後にIブロックを作ります。うまく動いていることが確認できたでしょうか。ブロックはこれで全てです。お疲れ様でした。

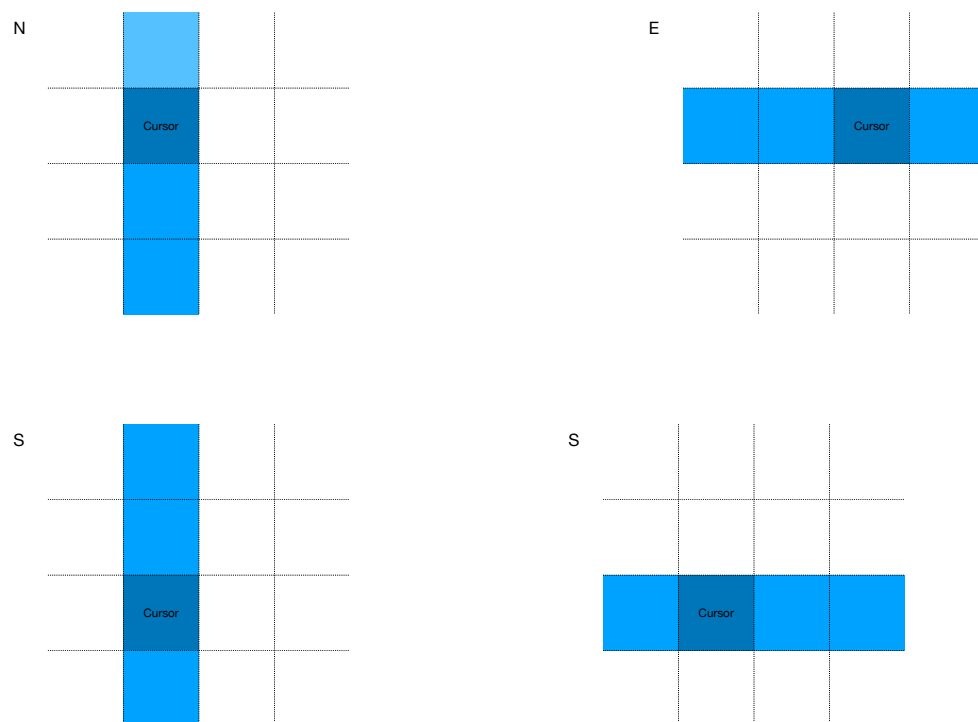


図 8.5 I ブロック

第 9 章

ブロックの落下とランダムなブロックの生成

9.1 ブロックの落下

今までのプログラムでは、ブロックが動くのはキー入力を受け取ったときだけでした。しかし、テトリスでは一定時間ごとにブロックが落ちてくるようになっています。今回は 1 秒に一度ブロックが落ちるようにしましょう。

9.1.1 1 秒ごとに落とす

1 秒を測るためには、時間を計測する必要があります。今回、main 関数の while 文は 1 秒間に 60 回実行されています。つまり、1 秒を測るためには 60 回のループを数えればよいということです。

ソースコード 9.1 main 関数を変更する

```
1 def main():
2     """省略"""
3
4     # chapter 9 ブロックを落とす
5     # ループ回数を数える変数を追加
6     # 時間にかかわるものの名前はtimer になることが多いですね
7     timer = 0
8
9     while True:
10        # timer を 1 増やします
11        """この行を消して書いてみてください"""
12
```

```
13     # Draw the board
14     board.draw(screen)
15
16     # Handle events
17     for event in pygame.event.get():
18         if event.type == pygame.QUIT:
19             return
20     # key handling
21     keys = pygame.key.get_pressed()
22     if keys[pygame.K_LEFT]:
23         board.cursor_move_left()
24         pygame.time.wait(100)
25     if keys[pygame.K_RIGHT]:
26         board.cursor_move_right()
27         pygame.time.wait(100)
28     if keys[pygame.K_DOWN]:
29         board.cursor_move_down()
30         pygame.time.wait(100)
31
32     # chapter 9 ブロックを落とす
33     # 一定時間ごとに下に動かします
34     # もし、timer が 60 になったら下に動かし、リセットもします
35     if timer == 60:
36         board.cursor_move_down()
37         timer = 0
38
39     """あとは同じです"""
40     # Update the display
41     pygame.display.update()
42
43     # Tick
44     # 今回while ループが 1 秒に 60 回回るようにしています
45     clock.tick(60)
46
47 if __name__ == "__main__":
48     main()
```

これを実行すると、1 秒ごとにブロックが落ちていくようになります。

9.1.2 ブロックを落とす

Board クラスに drop 関数を追加します。drop とは、落とす、こぼす、という意味があります。落ちれなくなるところまで落とすという関数です。

ソースコード 9.2 Board クラスに drop 関数を追加する

```
1 class Board:
2     """省略"""
3     """一番下あたりに書いてください"""
4     def drop(self):
5         while """ブロックが下に動ける""":
6             """下に動かすコード"""
```

また、main 関数を変更して、スペースキーでブロックを落とせるようにします。

ソースコード 9.3 main 関数を変更する

```
1 def main():
2     # Create the board
3     board = Board(tile_size=30)
4
5     # Initialize pygame
6     pygame.init()
7     screen = pygame.display.set_mode(board.window_size())
8     pygame.display.set_caption("Tetris")
9     clock = pygame.time.Clock()
10
11     board.moving_block = TBlock()
12
13     while True:
14         """省略"""
15         # key handling
16         keys = pygame.key.get_pressed()
17         if keys[pygame.K_LEFT]:
18             board.cursor_move_left()
19             pygame.time.wait(100)
20         if keys[pygame.K_RIGHT]:
21             board.cursor_move_right()
22             pygame.time.wait(100)
23         if keys[pygame.K_DOWN]:
24             board.cursor_move_down()
```

```

25         pygame.time.wait(100)
26     if keys[pygame.K_r]:
27         board.block_rotate()
28         pygame.time.wait(100)
29
30     # この辺に書き足します
31     # chapter 9 ブロックを落とす
32     if keys[pygame.K_SPACE]:
33         board.drop()
34         pygame.time.wait(100)

```

きちんと動いているでしょうか。

9.2 ランダムなブロックの生成

今度はランダムにブロックを生成する機能を追加します。generate_block 関数を作り、その中でランダムにブロックを生成するようにします*1。その関数は__init__関数の中で呼び出すと、最初のブロックがランダムになります。

ソースコード 9.4 ランダムなブロックの生成

```

1 from random import randint
2
3 """省略"""
4
5 class Board:
6     def __init__(self):
7         """省略"""
8         # chapter 9 ブロックを生成する で追加
9         # ブロックを生成します
10        self.generate_block()
11
12    """省略"""
13
14    """一番下あたりに書いてください"""
15    def generate_block(self):
16        # 1に戻すのは、上にはみ出るブロックのためです
17        """カーソルの y 座標を 1 に戻します"""
18        # ランダムにブロックを生成します

```

*1 generate: 生成する


```
19         block_type = randint(0, 6)
20         if block_type == 0:
21             self.moving_block = TBlock()
22         elif block_type == 1:
23             self.moving_block = OBlock()
24         elif block_type == 2:
25             self.moving_block = LBlock()
26         elif block_type == 3:
27             self.moving_block = JBlock()
28         elif block_type == 4:
29             self.moving_block = ZBlock()
30         elif block_type == 5:
31             self.moving_block = SBlock()
32         elif block_type == 6:
33             self.moving_block = IBlock()
```

これで、ランダムなブロックが生成されるようになりました。main でブロックを設定する必要がなくなったので、消してしまいましょう。

ソースコード 9.5 main 関数の変更

```
1 def main():
2     # Create the board
3     board = Board(tile_size=30)
4
5     # Initialize pygame
6     pygame.init()
7     screen = pygame.display.set_mode(board.window_size())
8     pygame.display.set_caption("Tetris")
9     clock = pygame.time.Clock()
10
11     # chapter 9 ブロックを生成する
12     # ここを消してしまいます。
13     # 3行目のBoard()の中で生成しているので、ここで生成する必要がなくなり
14     # ます。
15     board.moving_block = TBlock()
16
17     while True:
18         """省略"""
19         # key handling
20         keys = pygame.key.get_pressed()
```

```
20         if keys[pygame.K_LEFT]:
21             board.cursor_move_left()
22             pygame.time.wait(100)
23         if keys[pygame.K_RIGHT]:
24             board.cursor_move_right()
25             pygame.time.wait(100)
26         if keys[pygame.K_DOWN]:
27             board.cursor_move_down()
28             pygame.time.wait(100)
29         if keys[pygame.K_r]:
30             board.block_rotate()
31             pygame.time.wait(100)
32
33         if keys[pygame.K_SPACE]:
34             board.drop()
35             pygame.time.wait(100)
```

これで、ランダムなブロックが生成されるようになりました。しかし、ブロックが一番下まで落ちても、次のブロックに切り替わりません。次の章でその機能を追加します。

第 10 章

次のブロックへ切り替える

10.1 ブロックが一番下まで落ちた時とは

ブロックがこれ以上下に落ちられないとき、次のブロックに切り替える処理を書きます。現在、1 秒に一回、ブロックが落ちるようになっているので、そこを利用しましょう。

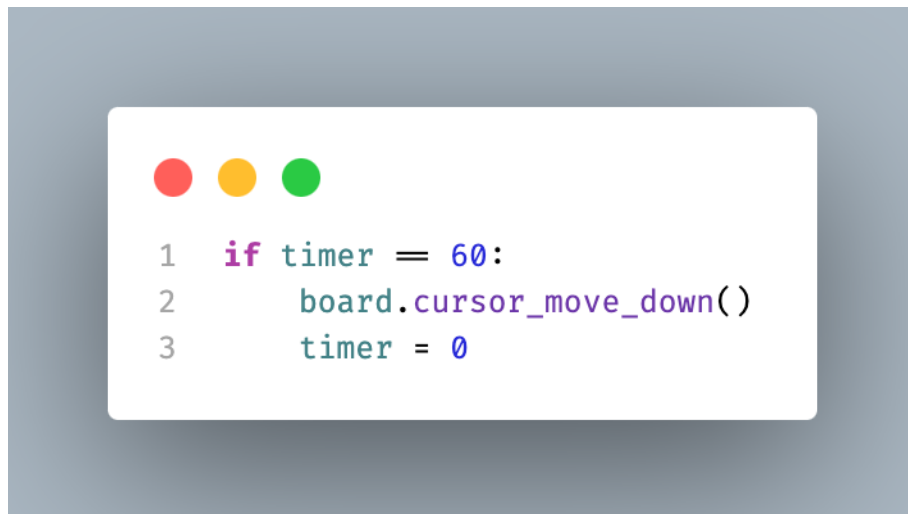


図 10.1 「一秒経った時」を実現している if 文

10.1.1 Board クラスに新しい関数 update を追加

Board クラスに新しい関数 update を追加します。main に「もしブロックが…」のような処理を書いても動きはするのですが、役割分担の考え方から Board クラスに書くことにします。

ソースコード 10.1 定期的にブロックを落とし、落とせないなら次のブロックを用意する update 関数

```

1 class Board:
2     """省略"""
3     def update(self):
4         # 下に動けるなら
5         if self.moving_block.can_go_down(self.cursor, self.board):
6             # 下に動かす
7             self.cursor.move_down(self.moving_block, self)
8         else:
9             # 下に動けないなら
10            # 新しくブロックを生成する
11            self.generate_block()

```

次に、main 関数を変更します。今まで単純にブロックを落としていた部分を update 関数に変更します。

ソースコード 10.2 main 関数を変更する

```

1
2 def main():
3     """省略"""
4     # chapter 10
5     # ブロックを生成する処理
6     if timer == 60:
7         # board.cursor_move_down()から変更
8         # 下に動かすだけでなく、動けない場合は新しいブロックを生成する
9         # ようになります
10        board.update()
11        timer = 0
12        # Update the display
13        pygame.display.update()
14
15        # Tick
16        clock.tick(60)
17
18 if __name__ == "__main__":
19     main()

```

実行すると、ブロックが一番下まで落ちたら次のブロックに切り替わるようになります。

10.2 ブロックを積む

前回のセクションでは、ブロックが切り替わった時に前のブロックが消えてしまう問題がありました。今回は、ブロックが一番下まで落ちた時に、そのブロックを置く処理を書きます。

10.2.1 Board クラスに fix 関数を追加

Board クラスに fix 関数を追加します^{*1}。ここで、盤面がどのように描かれているかを復習します。

スクリーンへの書き込みを行なっている場所

実際にウィンドウへの書き込みを行なっているのは、draw 関数です。その中でも、25 行目が盤面の描画、79 行目がカーソルを灰色にする処理、143 行目が現在持っているブロックの描画、242 行目が枠線の描画になっています。ここから、self.board の中身

```

1 def draw(self, screen):
2     for y in range(HEIGHT):
3         for x in range(WIDTH):
4             pygame.draw.rect(screen, self.board[y][x], (x * self.TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE, self.TILE_SIZE))
5
6     # Draw the cursor
7     for y in range(HEIGHT):
8         if self.board[y][self.cursor.x] == BLACK:
9             pygame.draw.rect(screen, CURSOR_COLOR, (self.cursor.x * self.TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE, self.TILE_SIZE))
10
11    # chapter6 ブロックを動かす
12    # ブロックを動かす処理を追加します
13    # 動かすブロックがNoneではない時
14    if self.moving_block is not None:
15        # ブロックの情報を取得します
16        # 座標がリストになって帰ってくるはずなので変数に入れておきます
17        datas = self.moving_block.block_info(self.cursor)
18
19        # 何色で塗るかblockのcolorという変数に入っています
20        # 一旦変数にコピーしておきます
21        color = self.moving_block.color
22
23        # ブロックからもらった座標リストを一つずつ見ていきます
24        # リストを一つずつ見ていくループはfor文を使います
25        for data in datas:
26            # for文で決めた変数に座標が一つずつ入ってループされます
27            # それぞれの座標の場所を塗ることが目標です
28
29            # その0番目にx座標が入っているのでそれをxに入れます
30            # 1番目にy座標が入っているのでそれをyに入れます
31            x = data[0]
32            y = data[1]
33
34            # マスを塗りつぶします。この行はわからなくて大丈夫ですが
35            # 色を入れた変数の名前を置き換えてください
36            pygame.draw.rect(screen, color, (x * self.TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE, self.TILE_SIZE))
37
38    # あとは前回と同じです
39    # 順番を間違えるとブロックが線を上書きして表示されてしまうので
40    # この順番です。書き足す位置を間違えないように気をつけてください
41    # Draw the grid
42    for y in range(HEIGHT):
43        pygame.draw.line(screen, GRAY, (0, y * self.TILE_SIZE), (WIDTH * self.TILE_SIZE, y * self.TILE_SIZE))
44    for x in range(WIDTH):
45        pygame.draw.line(screen, GRAY, (x * self.TILE_SIZE, 0), (x * self.TILE_SIZE, HEIGHT * self.TILE_SIZE))

```

図 10.2 draw 関数の中身

^{*1} fix: 固定する、直す

を変更すれば、draw 関数が自動的に画面に反映してくれそうです。



図 10.3 self.board を変更すれば画面にも反映されそう

10.2.2 fix 関数を実装

fix 関数を実装します。

ソースコード 10.3 fix 関数

```
1 class Board:
2     """省略"""
3     """一番下あたりに書いてください"""
4     def fix(self):
5         """ブロックの情報を board に書き込んでいきます"""
6         block_info = self.moving_block.get_block_info(self.cursor)
7         block_color = self.moving_block.color
8         for pos in block_info:
9             """pos には、(x, y) の形で座標が入っています"""
10            """board には [y][x] の形でアクセスしないといけないので、
11               pos を逆にしています"""
12            self.board[pos[1]][pos[0]] = block_color
```

次に、update 関数を変更します。

ソースコード 10.4 update 関数を変更

```
1 class Board:
2     """省略"""
3     def update(self):
4         # 下に動けるなら
5         if self.moving_block.can_go_down(self.cursor, self.board):
6             # 下に動かす
7             self.cursor.move_down(self.moving_block, self)
8         else:
9             # ここに追加
```

```
10         # ブロックを変えてしまう前に置いておきます
11         self.fix()
12         # 新しくブロックを生成する
13         self.generate_block()
```

実行すると、ブロックが一番下まで落ちたら、そのブロックが盤面に固定されるようになります。