

# Tetris

JiaSheng Yun

July 2023

## 1 Creating assets

### 1.1 use string and vector to create object

---

```
1 # creating assets
2 tetromino = []
3 object1 = "..X.\n" \
4           "..X.\n" \
5           "..X.\n" \
6           "..X.\n"
7 object2 = "..X.\n" \
8           ".XX.\n" \
9           ".X..\n" \
10          "....\n"
11 object3 = ".X..\n" \
12          ".XX.\n" \
13          "..X.\n" \
14          "....\n"
15 object4 = "....\n" \
16          ".XX.\n" \
17          ".XX.\n" \
18          "....\n"
19 object5 = "..X.\n" \
20          ".XX.\n" \
21          "..X.\n" \
22          "....\n"
23 object6 = "....\n" \
24          ".XX.\n" \
25          "..X.\n" \
26          "..X.\n"
27 object7 = "....\n" \
28          ".XX.\n" \
29          ".X..\n" \
30          ".X..\n"
31
32 tetromino.append(object1)
33 tetromino.append(object2)
```

```

34 tetromino.append(object3)
35 tetromino.append(object4)
36 tetromino.append(object5)
37 tetromino.append(object6)
38 tetromino.append(object7)

```

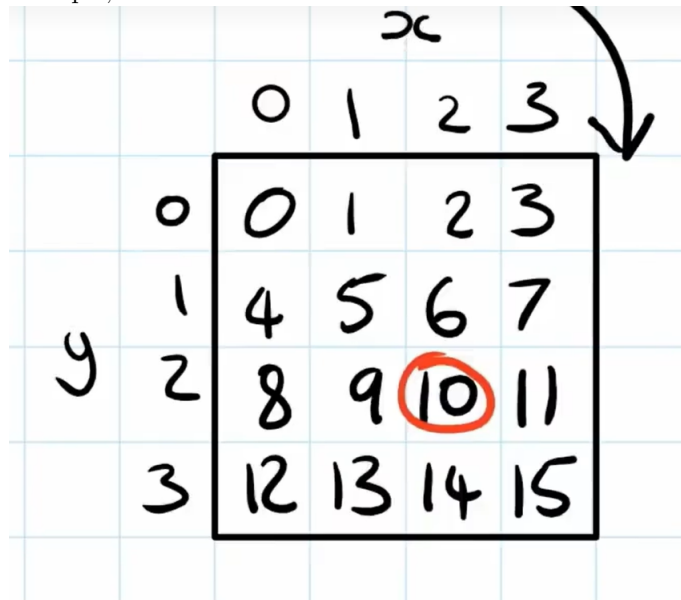
---

## 2 Rotating blocks

We often hope to rotate blocks.

### 2.1 use one-dimensional vectors to represent two-dimensional vectors

for example, we have a two-dimensional vector like this:



It is easy for us to index content by  $x$  and  $y$  ( $a[x][y]$ ). We can also use one-dimensional vector to index the content if we add a numerical label to each position. For instance, we can index the position  $a[2][2]$  by  $i[10]$ .  $10 = 2*4+2$  which means  $y*width+x$ .

### 2.2 rotating

If we rotate the block  $90^\circ$ , we can obtain the following figure

			$x$			
			0	1	2	3
	0		12	8	4	0
	1		13	9	5	1
y	2		14	10	6	2
	3		15	11	7	3

if we still want to index the position  $a[2][2]$ , we should index it by  $i[6]$  which is easy to get from the figure.

It can be easily seen through the pictures that  $x+1$  number size -4 and  $y+1$  number size +1. Therefore, the index number is  $12-4x+y$ .

Similarly, it can be concluded that:

				$x$			
			0	1	2	3	
	0	15	14	13	12		
	1	11	10	9	8		
$y$	2	7	6	5	4		
	3	3	2	1	0		

If I rotate the block  $180^\circ$ , the index number is  $15-4y-x$

			0	1	2	3	
	0	3	7	11	15		
	1	2	6	10	14		
$y$	2	1	5	9	13		
	3	0	4	8	12		

if I rotate the block  $270^\circ$ , the index number is  $3-y+4x$

```

1 # rotate
2 def rotate(px, py, r):
3     if r == 0:
4         return py*4+px
5     elif r == 1:
6         return 12+py-4*px
7     elif r == 2:
8         return 15-4*py-px
9     elif r == 3:
10        return 3-py+4*px

```

---

### 3 Field

In this section, I will construct a 12\*18 Field. And I will use 9 to represent border.

```

1 # Field
2 fieldwidth = 12
3 fieldheight = 18
4
5 pField = []
6 for x in range(fieldwidth):
7     for y in range(fieldheight):
8         pField[y*fieldwidth+x] = 9 if (x == 0 or x == fieldwidth-1 or y
9             == fieldheight-1) else 0

```

---

In this project, I use pygame to implement site construction.

```

1 import pygame
2 # game window
3 pygame.init()
4 window_width = 480
5 window_height = 720
6 window_title = "Tetris"
7
8 screen = pygame.display.set_mode((window_width, window_height))
9 pygame.display.set_caption(window_title)
10
11 running = True
12 while running:
13     for event in pygame.event.get():
14         if event.type == pygame.QUIT:
15             running = False
16
17     # fill in black background
18     screen.fill((0, 0, 0))
19     # implement the game interface and blocks here
20     for x in range(fieldwidth):

```

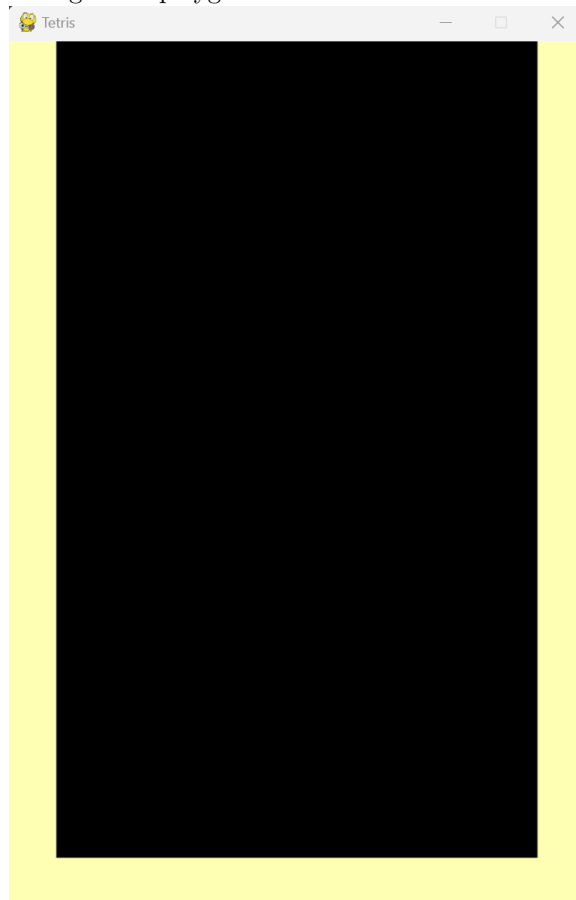
```

21     for y in range(fieldheight):
22         # print(y*fieldwidth+x)
23         if pField[y*fieldwidth+x] == 9:
24             pygame.draw.rect(screen, (255,255,255), (x*40, y*40, 40,
25                                     40))
26         else:
27             ...
28     pygame.display.update()
29     pygame.quit()

```

---

Now we get the playground:



## 4 Game Loop

Game loop is the most important part of game engine. These are the sequences that everything is going on. Simple games like Tetris are not large-scale event driven applications. It includes some elements like timing, user input, dating

the game logic and then draw it on the screen. It keep doing this until the game is over or user exits it.

#### 4.1 Determine if the position is appropriate

This function has four parameters:

nTetromino: it represents the type of the block

nrotation: it represents rotational angle

positionX: it represents the horizontal coordinates of fourth order blocks

positionY: it represents the vertical coordinates of fourth order blocks

---

```
1 # judge if the position of the piece fit the rule
2 def doesPieceFit(nTetromino, nrotation, positionX, positionY):
3     for px in range(4):
4         for py in range(4):
5             # get the index
6             pi = rotate(px, py, nrotation)
7
8             #get index into field
9             fi = (positionY+py)*12+positionX+px
10            if (positionX+px >= 0) and (positionX+px <= 11):
11                if (positionY+py >= 0) and (positionY+py <= 18):
12                    if tetromino[nTetromino][pi] == 'X' and ((pField[fi]
13                        == 1) or (pField[fi] == 9)):
14                        return False
15
16            return True
```

---

#### 4.2 Inputs and corresponding operations

There are four types of inputs in Tetris: left shift, right shift, down shift, and rotation. In the program, I use the left, right, down, and z keys to correspond to four different operations.

---

```
1 for event in pygame.event.get():
2     if event.type == pygame.QUIT:
3         gameover = True
4     elif event.type == pygame.KEYDOWN:
5         if event.key == pygame.K_LEFT:
6             bKey = 1
7         elif event.key == pygame.K_RIGHT:
8             bKey = 2
9         elif event.key == pygame.K_DOWN:
10            bKey = 3
11        elif event.key == pygame.K_z:
12            bKey = 4
13    # Game logic =====
```

```

14     if bKey == 1 and doesPieceFit(nCurrentPiece, nCurrentRotation,
15         nCurrentX-1, nCurrentY):
16         nCurrentX = nCurrentX - 1
17         bKey = 0
18
19     if bKey == 2 and doesPieceFit(nCurrentPiece, nCurrentRotation,
20         nCurrentX+1, nCurrentY):
21         nCurrentX = nCurrentX + 1
22         bKey = 0
23
24     if bKey == 3 and doesPieceFit(nCurrentPiece, nCurrentRotation,
25         nCurrentX, nCurrentY+1):
26         nCurrentY = nCurrentY + 1
27         bKey = 0
28
29     if bKey == 4 and doesPieceFit(nCurrentPiece, (nCurrentRotation+1)%4,
30         nCurrentX, nCurrentY):
31         nCurrentRotation += 1
32         nCurrentRotation %= 4
33         bKey = 0

```

---

However, if the above method is used, only one button can be input per round, so I used a list to save the input for each round.

---

```

1  bKey = []
2  clock = pygame.time.Clock()
3  fps = 60
4  wait_time = 100
5  # Render output =====
6
7  while not gameover:
8      # game timing =====
9      clock.tick(fps)
10     # input =====
11     for event in pygame.event.get():
12         if event.type == pygame.QUIT:
13             gameover = True
14         elif event.type == pygame.KEYDOWN:
15             if event.key == pygame.K_LEFT:
16                 # bKey = 1
17                 bKey.append(1)
18             elif event.key == pygame.K_RIGHT:
19                 # bKey = 2
20                 bKey.append(2)
21             elif event.key == pygame.K_DOWN:
22                 # bKey = 3
23                 bKey.append(3)
24             elif event.key == pygame.K_z:
25                 # bKey = 4

```



```

26         bKey.append(4)
27     # Game logic =====
28     if (1 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation,
29         nCurrentX-1, nCurrentY):
30         nCurrentX = nCurrentX - 1
31         bKey.remove(1)
32         # bKey=0
33
34     if (2 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation,
35         nCurrentX+1, nCurrentY):
36         nCurrentX = nCurrentX + 1
37         bKey.remove(2)
38         # bKey=0
39
40     if (3 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation,
41         nCurrentX, nCurrentY+1):
42         nCurrentY = nCurrentY + 1
43         bKey.remove(3)
44         # bKey=0
45
46     if (4 in bKey) and doesPieceFit(nCurrentPiece,
47         (nCurrentRotation+1)%4, nCurrentX, nCurrentY):
48         nCurrentRotation += 1
49         nCurrentRotation %= 4
50         bKey.remove(4)
51         # bKey=0

```

---

### 4.3 Block drop

In Tetris, blocks fall down periodically. The dropping function can be easily achieved by adding a time judgment.

```

1     fall_time = 0.5
2     last_fall_time = time.time()
3     if current_time - last_fall_time >= fall_time:
4         if doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX,
5             nCurrentY + 1):
6             nCurrentY += 1
7             last_fall_time = current_time

```

---

Of course, we also need to add a judgement function to see if the block can continue to fall. If it cannot fall, we need to complete a sequence of things.

First, we need to lock the current piece into the field. Then, check if we get any lines. Finally, we choose next piece and check if piece fits.

```

1     else:
2         # lock the current piece into the field.
3         for px in range(4):

```

---

```

4         for py in range(4):
5             if tetromino[nCurrentPiece][rotate(px, py,
6                 nCurrentRotation)] == 'X':
7                 # print(len(pField))
8                 #
9                 print(int((nCurrentY+py)*fieldwidth+nCurrentX+px))
10                pField[int((nCurrentY+py)*fieldwidth+nCurrentX+px)]
11                    = 1
12
13            # check if we get any lines.
14            for py in range(4):
15                if nCurrentY+py<fieldheight-1:
16                    bline = True
17                    for px in range(1, fieldwidth):
18                        if pField[(nCurrentY+py)*fieldwidth+px] == 0:
19                            bline = False
20                    if bline:
21                        # remove the line
22                        for px in range(1, fieldwidth-1):
23                            pField[(nCurrentY + py) * fieldwidth + px] = 8
24                        vline.append(nCurrentY+py)
25
26            # Finally, we choose next piece
27            nCurrentX = fieldwidth//2
28            nCurrentY = 0
29            nCurrentRotation = 0
30            nCurrentPiece = random.randint(0, 6)
31            # check if piece fits.
32            gameover = not doesPieceFit(nCurrentPiece, nCurrentRotation,
33                nCurrentX, nCurrentY)

```

---

```

1  if len(vline) != 0:
2      pField_copy = copy.deepcopy(pField)
3      for dy in vline:
4          for dx in range(1, fieldwidth-1):
5              pField[dy*fieldwidth+dx] = 0
6      for dy in range(0, vline[0]):
7          for dx in range(1, fieldwidth-1):
8              pField[(dy + len(vline)) * fieldwidth + dx] = 0
9      for dy in range(0, vline[0]):
10         for dx in range(1, fieldwidth-1):
11             if pField_copy[dy*fieldwidth+dx] == 1:
12                 pField[(dy+len(vline))*fieldwidth+dx] = 1
13         vline = []

```

---

## 5 Additional features: increase difficulty and score display

It is easy to implement.

---

```
1 nPieceCount = 1
2 nscore = 0
3
4     for py in range(4):
5         if nCurrentY+py<fieldheight-1:
6             bline = True
7             for px in range(1, fieldwidth):
8                 if pField[(nCurrentY+py)*fieldwidth+px] == 0:
9                     bline = False
10            if bline:
11                # remove the line
12                for px in range(1, fieldwidth-1):
13                    pField[(nCurrentY + py) * fieldwidth + px] = 8
14                vline.append(nCurrentY+py)
15            # Finally, we choose next piece
16            nCurrentX = fieldwidth//2
17            nCurrentY = 0
18            nCurrentRotation = 0
19            nCurrentPiece = random.randint(0, 6)
20            nPieceCount += 1
21            nscore += 100
22
23 font = pygame.font.Font(None, 36)
24 score_text = font.render("Score: " + str(nscore), True, (255, 135,
25                               207))
25 screen.blit(score_text, (10, 10))
```

---