Tetris

JiaSheng Yun

July 2023

1 Creating assets

1.1 use string and vector to create object

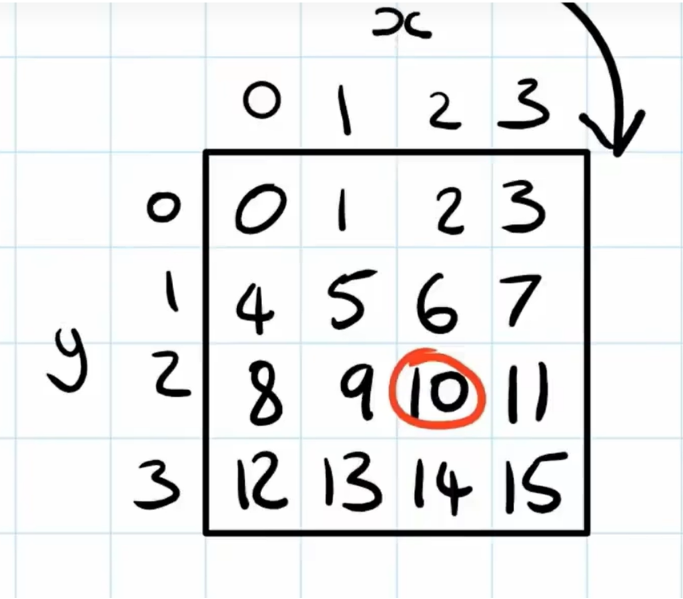
```
# creating assets
tetromino = []
object1 = "..X.\n" \
          "..X.\n" \
          "..X.\n" \
          "..X.\n"
object2 = "..X.\n" \
          ".XX.\n" \
          ".X..\n" \
          "....\n"
object3 = ".X..\n" \
          ".XX.\n" \
          "..X.\n" \
          "....\n"
object4 = "....\n" \
          ".XX.\n" \
          ".XX.\n" \
          "....\n"
object5 = "..X.\n" \
          ".XX.\n" \
          "..X.\n" \
          "....\n"
object6 = "....\n" \
          ".XX.\n" \
          "..X.\n" \
          "..X.\n"
object7 = "....\n" \
          ".XX.\n" \
          ".X..\n" \
          ".X..\n"

tetromino.append(object1)
tetromino.append(object2)
tetromino.append(object3)
tetromino.append(object4)
tetromino.append(object5)
tetromino.append(object6)
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:

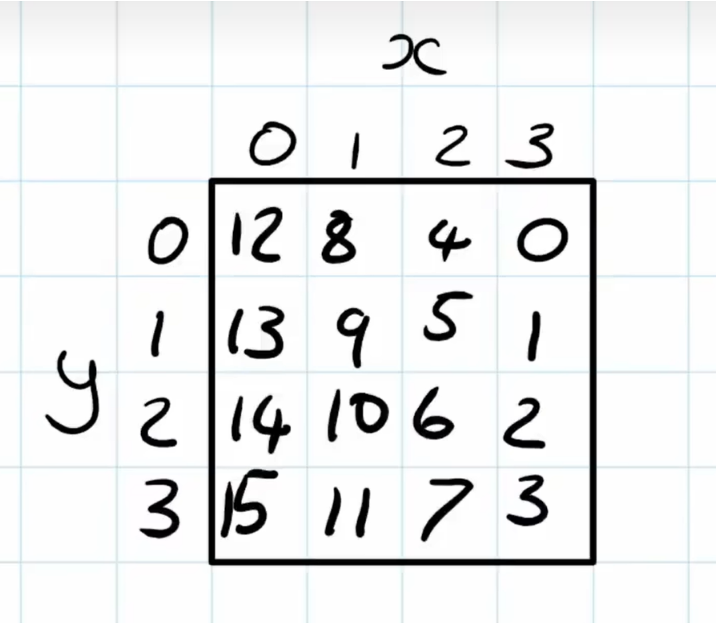


	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

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°, we can obtain the following figure

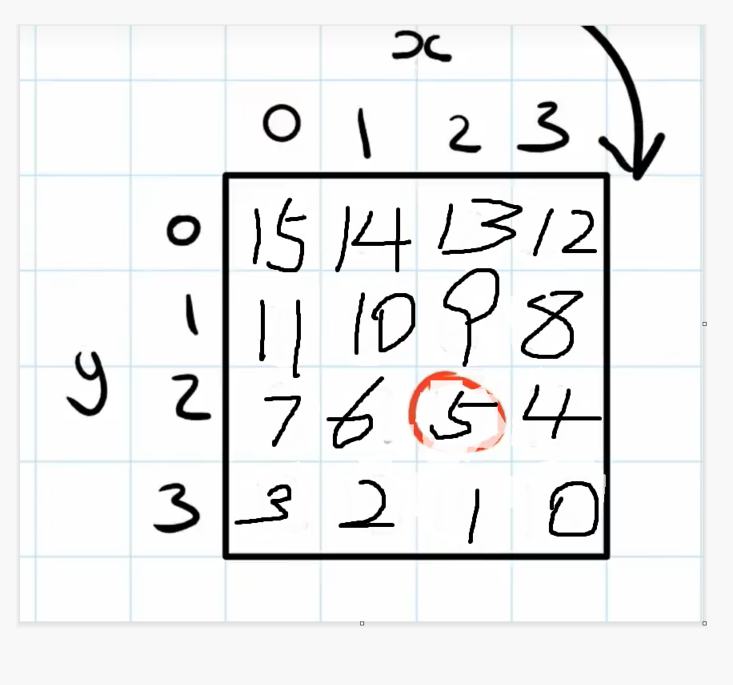


	0	1	2	3
0	12	8	4	0
1	13	9	5	1
2	14	10	6	2
3	15	11	7	3

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.

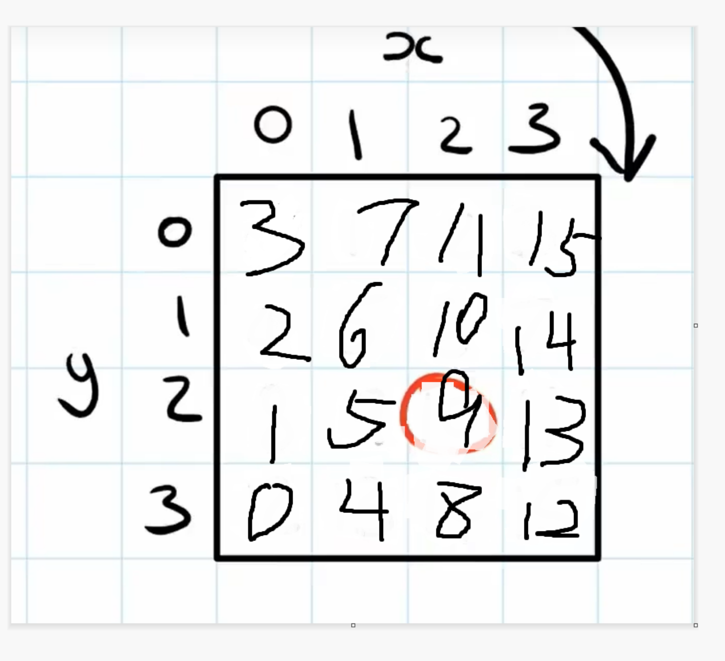
It can be easily seen through the pictures that x+1 number size -4 and y+1 number size +1. Therefore, the index number is 12-4x+y.

Similarly, it can be concluded that:



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

If I rotate the block 180°, the index number is 15-4y-x



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

if I rotate the block 270°, the index number is 3-y+4x

```python
# rotate
def rotate(px, py, r):
    if r == 0:
        return py*4+px
    elif r == 1:
        return 12+py-4*px
    elif r == 2:
        return 15-4*py-px
    elif r == 3:
        return 3-py+4*px
```

3 Field

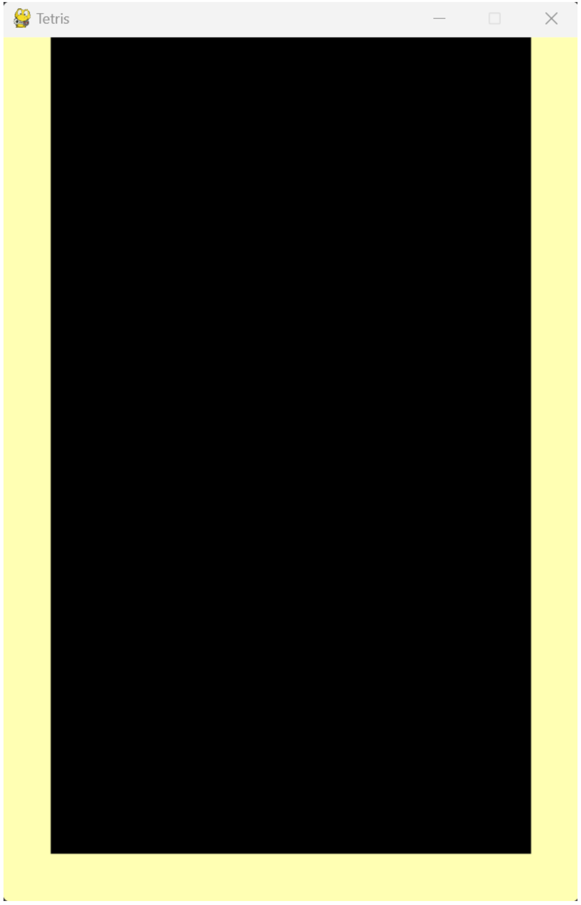
In this section, I will construct a 12*18 Field. And I will use 9 to represent border.

```python
# Field
fieldwidth = 12
fieldheight = 18

pField = []
for x in range(fieldwidth):
    for y in range(fieldheight):
        pField[y*fieldwidth+x] = 9 if (x == 0 or x == fieldwidth-1 or y
             == fieldheight-1) else 0
```

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

```python
import pygame
# game window
pygame.init()
window_width = 480
window_height = 720
window_title = "Tetris"

screen = pygame.display.set_mode((window_width, window_height))
pygame.display.set_caption(window_title)

running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    # fill in black background
    screen.fill((0, 0, 0))
    # implement the game interface and blocks here
    for x in range(fieldwidth):
        for y in range(fieldheight):
            # print(y*fieldwidth+x)
            if pField[y*fieldwidth+x] == 9:
                pygame.draw.rect(screen, (255,255,255), (x*40, y*40, 40,
                    40))
            else:
                ...
    pygame.display.update()
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

```
# judge if the position of the piece fit the rule
def doesPieceFit(nTetromino, nrotation, positionX, positionY):
    for px in range(4):
        for py in range(4):
            # get the index
            pi = rotate(px, py, nrotation)

            #get index into field
            fi = (positionY+py)*12+positionX+px
            if (positionX+px >= 0) and (positionX+px <= 11):
                if (positionY+py >= 0) and (positionY+py <= 18):
                    if tetromino[nTetromino][pi] == 'X' and ((pField[fi]
                        == 1) or (pField[fi] == 9)):
                        return False

    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.

```
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            gameover = True
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                bKey = 1
            elif event.key == pygame.K_RIGHT:
                bKey = 2
            elif event.key == pygame.K_DOWN:
                bKey = 3
            elif event.key == pygame.K_z:
                bKey = 4
    # Game logic ======================================================
    if bKey == 1 and doesPieceFit(nCurrentPiece, nCurrentRotation,
        nCurrentX-1, nCurrentY):
        nCurrentX = nCurrentX - 1
        bKey = 0

    if bKey == 2 and doesPieceFit(nCurrentPiece, nCurrentRotation,
        nCurrentX+1, nCurrentY):
        nCurrentX = nCurrentX + 1
        bKey = 0

    if bKey == 3 and doesPieceFit(nCurrentPiece, nCurrentRotation,
        nCurrentX, nCurrentY+1):
        nCurrentY = nCurrentY + 1
        bKey = 0

    if bKey == 4 and doesPieceFit(nCurrentPiece, (nCurrentRotation+1)%4,
        nCurrentX, nCurrentY):
        nCurrentRotation += 1
        nCurrentRotation %= 4
        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.

```
bKey = []
clock = pygame.time.Clock()
fps = 60
wait_time = 100
# Render output ===================================================

while not gameover:
    # game timing =====================================================
    clock.tick(fps)
    # input ===========================================================
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            gameover = True
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                # bKey = 1
                bKey.append(1)
            elif event.key == pygame.K_RIGHT:
                # bKey = 2
                bKey.append(2)
            elif event.key == pygame.K_DOWN:
                # bKey = 3
                bKey.append(3)
            elif event.key == pygame.K_z:
                # bKey = 4
                bKey.append(4)
    # Game logic ======================================================
    if (1 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX-1, nCurrentY):
        nCurrentX = nCurrentX - 1
        bKey.remove(1)
        # bKey=0

    if (2 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX+1, nCurrentY):
        nCurrentX = nCurrentX + 1
        bKey.remove(2)
        # bKey=0

    if (3 in bKey) and doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX, nCurrentY+1):
        nCurrentY = nCurrentY + 1
        bKey.remove(3)
        # bKey=0

    if (4 in bKey) and doesPieceFit(nCurrentPiece, (nCurrentRotation+1)%4, nCurrentX, nCurrentY):
        nCurrentRotation += 1
        nCurrentRotation %= 4
        bKey.remove(4)
        # bKey=0
```

4.3 Block drop

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

```
    fall_time = 0.5
    last_fall_time = time.time()
    if current_time - last_fall_time >= fall_time:
        if doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX, nCurrentY + 1):
            nCurrentY += 1
            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.

```
        else:
            # lock the current piece into the field.
            for px in range(4):
                for py in range(4):
                    if tetromino[nCurrentPiece][rotate(px, py, nCurrentRotation)] == 'X':
                        # print(len(pField))
                        # print(int((nCurrentY+py)*fieldwidth+nCurrentX+px))
                        pField[int((nCurrentY+py)*fieldwidth+nCurrentX+px)] = 1
            # check if we get any lines.
            for py in range(4):
                if nCurrentY+py<fieldheight-1:
                    bline = True
                    for px in range(1, fieldwidth):
                        if pField[(nCurrentY+py)*fieldwidth+px] == 0:
                            bline = False
                    if bline:
                        # remove the line
                        for px in range(1, fieldwidth-1):
                            pField[(nCurrentY + py) * fieldwidth + px] = 8
                        vline.append(nCurrentY+py)
            # Finally, we choose next piece
            nCurrentX = fieldwidth//2
            nCurrentY = 0
            nCurrentRotation = 0
            nCurrentPiece = random.randint(0, 6)
            # check if piece fits.
            gameover = not doesPieceFit(nCurrentPiece, nCurrentRotation, nCurrentX, nCurrentY)
```

```
    if len(vline) != 0:
        pField_copy = copy.deepcopy(pField)
        for dy in vline:
            for dx in range(1, fieldwidth-1):
                pField[dy*fieldwidth+dx] = 0
        for dy in range(0, vline[0]):
            for dx in range(1, fieldwidth-1):
                pField[(dy + len(vline)) * fieldwidth + dx] = 0
        for dy in range(0, vline[0]):
            for dx in range(1, fieldwidth-1):
                if pField_copy[dy*fieldwidth+dx] == 1:
                    pField[(dy+len(vline))*fieldwidth+dx] = 1
        vline = []
```