

Tetris v pygame

Tento dokument je určený pro začátečníky/středně pokročilé programátory na střední škole. Veškeré funkce jsou popsány v kódu.

Tetromina, neboli kostky skládající se ze 4 čtverečků, padají po obrazovce a hráč je rovná do zdi od dolního konce hrací plochy. Řádek zaplněný bez děr zmizí. Hráč se snaží co nejdéle „odmazávat“ řádky; hra končí, když zeď dostoupí horního okraje hrací plochy

Začneme třídou kostky. Naším cílem je ukládat typy postav spolu s rotacemi.

```
class Figure:
    x = 0
    y = 0

    # tvary kostiček a jejich rotace, 0 = 0°, 1 = 90°, 2 = 180°, 3 = 270°
    figures = [
        [[1, 5, 9, 13], [4, 5, 6, 7], [1, 5, 9, 13], [4, 5, 6, 7]],
        [[4, 5, 9, 10], [2, 6, 5, 9], [4, 5, 9, 10], [2, 6, 5, 9]],
        [[6, 7, 9, 10], [1, 5, 6, 10], [6, 7, 9, 10], [1, 5, 6, 10]],
        [[1, 2, 5, 9], [0, 4, 5, 6], [1, 5, 9, 8], [4, 5, 6, 10]],
        [[1, 2, 6, 10], [5, 6, 7, 9], [2, 6, 10, 11], [3, 5, 6, 7]],
        [[1, 4, 5, 6], [1, 4, 5, 9], [4, 5, 6, 9], [1, 5, 6, 9]],
        [[1, 2, 5, 6], [1, 2, 5, 6], [1, 2, 5, 6], [1, 2, 5, 6]],
    ]
```

Kde hlavní seznam obsahuje typy obrázků a vnitřní seznamy obsahují jejich rotace. Čísla na každém obrázku představují pozice v matici 4x4, kde je obrázek plný. Například obrázek [1,5,9,13] představuje čáru.

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

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

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

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

Náhodně vybereme typ a barvu.

```
colors = [
    (0, 0, 0),
    (255, 0, 0),
    (0, 255, 0),
    (0, 0, 255)
]
```

A potřebujeme být schopni rychle otočit a získat aktuální rotaci figury, k tomu máme tyto dvě metody:

```
29 |
30 | def __init__(self, x, y):
31 |     self.x = x
32 |     self.y = y
33 |     self.type = random.randint(0, len(self.figures) - 1) # náhodný výběr tvaru kostičky
34 |     self.color = random.randint(1, len(colors) - 1) # výběr barvy pro tuto kostičku
35 |     self.rotation = 0 # výchozí rotace bude s indexem 0
36 |
37 | def image(self): # vrácení tvaru objektu s požadovanou rotací
38 |     return self.figures[self.type][self.rotation]
39 |
40 | def rotate(self): # otáčení objektu
41 |     self.rotation = (self.rotation + 1) % len(self.figures[self.type])
42 |
```

Nejprve inicializujeme hru pomocí několika proměnných:

```

class Tetris:
    level = 2 # obtížnost - vyšší číslo zvyšuje rychlost padání kostiček
    score = 0 # skóre, počítá se ze zničených plných řádků
    state = "start" # stav hry, poté z něho zjišťujeme, jestli jsme prohráli nebo stále hrajeme
    field = [] # hrací pole - bude obsahovat již položené kostičky tetrisu
    height = 0 # výchozí výška herního plánu
    width = 0 # výchozí šířka herního plánu
    x = 100 # umístění hracího plánu v okně na souřadnici X
    y = 60 # umístění hracího plánu v okně na souřadnici Y
    zoom = 20 # škálování velikosti hry
    figure = None

```

vytvoříme pole o velikosti výška x šířka.

```

def __init__(self, height, width):
    self.height = height
    self.width = width
    self.field = []
    self.score = 0
    self.state = "start"
    for i in range(height):
        new_line = []
        for j in range(width):
            new_line.append(0)
        self.field.append(new_line)

```

Vytvoření nové kostičky a její umístění na souřadnice (3,0)

```

def new_figure(self): #vytvoření nové kostičky
    self.figure = Figure(3, 0)

```

Další funkcí je kontrola, zda se právě letící postava protíná s něčím. To se může stát, když se postava pohybuje doleva, doprava, dolů nebo se otáčí.

```

def intersects(self): # metoda zjišťující, jestli kostička přesahuje výšku hracího
    intersection = False
    for i in range(4):
        for j in range(4):
            if i * 4 + j in self.figure.image():
                if i + self.figure.y > self.height - 1 or \
                    j + self.figure.x > self.width - 1 or \
                    j + self.figure.x < 0 or \
                    self.field[i + self.figure.y][j + self.figure.x] > 0:
                    intersection = True
    return intersection

```

zkontrolujeme každou buňku v matici 4x4 aktuální kostky, zda není mimo herní hranice a zda se nedotýká nějakého rušného herního pole. Zkontrolujeme, zda `self.field[..][..] > 0`, protože tam může být jakákoliv barva. A pokud je tam nula, znamená to, že pole je prázdné, takže není problém.

S touto funkcí nyní můžeme zkontrolovat, zda můžeme s kostičkami pohybovat nebo otáčet. Pokud se pohybuje dolů a protíná, znamená to, že jsme dosáhli dna, takže musíme „zmrazit“ kostičku na našem poli:

```

def break_lines(self): # Zničení řádku kostiček na herním plánu, když je celý řádek obsazen
    lines = 0
    for i in range(1, self.height):
        zeros = 0
        for j in range(self.width):
            if self.field[i][j] == 0:
                zeros += 1
        if zeros == 0:
            lines += 1
            for i1 in range(i, 1, -1):
                for j in range(self.width):
                    self.field[i1][j] = self.field[i1 - 1][j]
    self.score += lines ** 2

```

Po zmrazení musíme zkontrolovat, zda tam nejsou nějaké hotové řádky, které by měly být zničeny. Poté vytvoříme novou kostku, a pokud se již protíná, hra končí

Metoda pro pohyb po „hřišti“ s kostkami.

```
def go_space(self): # Metoda pro okamžité položení kostičky na nejnižší možnou souřadnici Y
    while not self.intersects():
        self.figure.y += 1
    self.figure.y -= 1
    self.freeze()

def go_down(self): # posun kostičky o jedno pole dolů
    self.figure.y += 1
    if self.intersects():
        self.figure.y -= 1
        self.freeze()

def freeze(self): # položení kostičky a zapamatování její pozice, přidání do herního plánu
    for i in range(4):
        for j in range(4):
            if i * 4 + j in self.figure.image():
                self.field[i + self.figure.y][j + self.figure.x] = self.figure.color
    self.break_lines()
    self.new_figure()
    if self.intersects():
        self.state = "gameover"

def go_side(self, dx): # posun kostičky do strany
    old_x = self.figure.x
    self.figure.x += dx
    if self.intersects():
        self.figure.x = old_x

def rotate(self): # otočení kostičky
    old_rotation = self.figure.rotation
    self.figure.rotate()
    if self.intersects():
```

Jak můžete vidět, metoda `go_space` prakticky duplikuje metodu `go_down`, ale klesá, dokud nedosáhne dna nebo nějaké již položené kostky.

A v každé metodě si pamatujeme poslední pozici, měníme souřadnice a kontrolujeme, jestli tam není průsečík. Pokud ano, vrátíme se do předchozího stavu.

Nyní zbývá už jen nastavit rozhraní v pygame.

```
# spuštění instance pygame
pygame.init()

# základní barvy využité pro okno s hrou
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GRAY = (128, 128, 128)

size = (400, 500) # výška a šířka našeho pygame okna
screen = pygame.display.set_mode(size)

pygame.display.set_caption("hra tetris") # nastavení názvu okna

done = False # proměnná, dle které budeme posuzovat, jestli se má hra vypnout nebo zůstat otevřená
clock = pygame.time.Clock()
fps = 20 # Dále v kodu tuto proměnnou využíváme pro výpočet rychlosti hry na základě levelu
game = Tetris(20, 10) # vytvoření herního plánu o výšce 20 kostiček, šířce 10 kostiček
counter = 0 # počítadlo ticků hry

pressing_down = False # Proměnná pro uložení stavu stisknutých kláves
```

Pro výběr barev ve hře nebo i změna barvy jednotlivých kostiček můžeme změnit pomocí rgb collors na <https://www.colorspire.com/rgb-color-wheel/>.