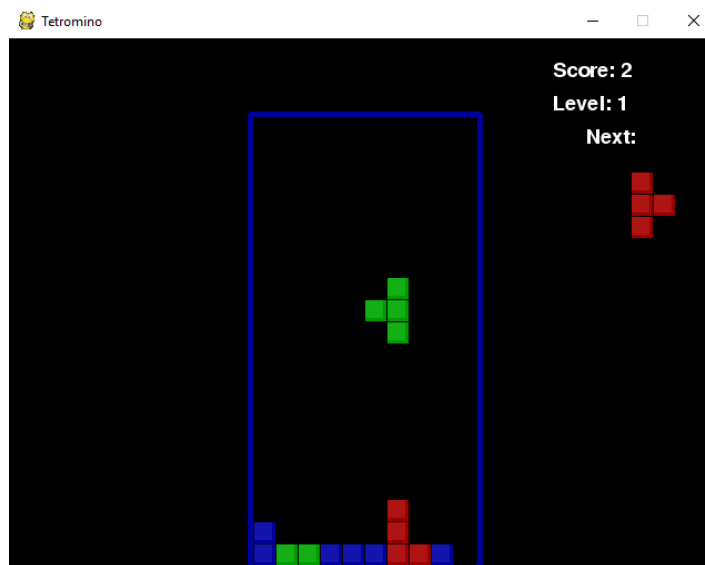


## Тетромино



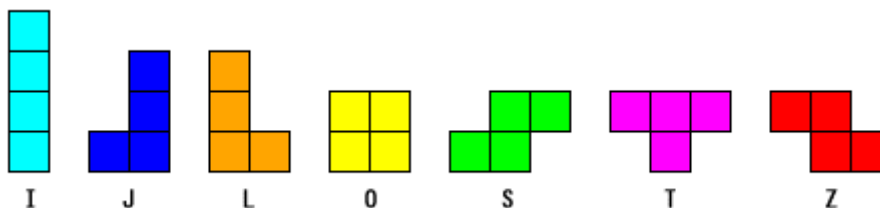
### Како играти Тетромино:

Тетромино је копија Тетриса. Различити облици блокова (сваки направљен од четири коцкица) падају са врха екрана и играч мора да их наводи како да се спуштају да би попунио ред без размака. Када је попуњен цео ред, ред је нестаје и сви редови изнад падају један ред наниже. Играч игра све док се не попуне сви редови и не постоји могућност да са врха падне нови блок.

### Неки од елемената Тетромина:

У овом поглављу, објаснићемо различите ствари програма за нашу игру.

- Табла(board) – табла је направљена од 10x20 слободних места на која смештамо наше блокове.
- Квадрат(box) – квадрат је поље четвртастог облика које се налази унутар табле.
- Део(piece) – ствари које падају са врха табле коју играч ротира и позиционира. Сваки део има свој облик и састављен од 4 квадрата.
- Облик(shape) – облици су различити типови делова у игрици. Назвали смо их I, J, L, O, S, T, Z.



- Шаблон(template) – листа која показује све могуће облике ротације. Ово чувамо у променљиве као што су S\_SHAPE\_TEMPLATE или J\_SHAPE\_TEMPLATE.
- Приземљен(landed) – када облик додирне дно или додирне неки други облик тада кажемо да је облик приземљен. У том тренутку почиње да пада нови облик са врха.

## Изворни код за Тетроминио

Изворни код можете скинути са <http://invpy.com/tetromino.py> . Можете да копирате и налепите код са линка <http://invpy.com/diff/tetromino> и упоредите са вашим кодом.

Требаћете да скинете музику и да ставите у исти фолдер са фајлом tetromino.py. Музику можете да скинете са наредних линкова:

- <http://invpy.com/tetrisb.mid>
- <http://invpy.com/tetrisc.mid>

```
1 # Tetromino (a Tetris clone)
2 # By Al Sweigart al@inventwithpython.com
3 # http://inventwithpython.com/pygame
4 # Released under a "Simplified BSD" license
5
6 import random, time, pygame, sys
7 from pygame.locals import *
8
9 FPS = 25
10 WINDOWWIDTH = 640
11 WINDOWHEIGHT = 480
12 BOXSIZE = 20
13 BOARDWIDTH = 10
14 BOARDHEIGHT = 20
15 BLANK = '.'
16
17 MOVESIDEWAYSFREQ = 0.15
18 MOVEDOWNFREQ = 0.1
19
20 XMARGIN = int((WINDOWWIDTH - BOARDWIDTH * BOXSIZE) / 2)
21 TOPMARGIN = WINDOWHEIGHT - (BOARDHEIGHT * BOXSIZE) - 5
22
23 #      R   G   B
24 WHITE  = (255, 255, 255)
25 GRAY   = (185, 185, 185)
26 BLACK  = ( 0,  0,  0)
27 RED    = (155,  0,  0)
28 LIGHTRED = (175, 20, 20)
29 GREEN  = ( 0, 155,  0)
30 LIGHTGREEN = ( 20, 175, 20)
31 BLUE   = ( 0,  0, 155)
32 LIGHTBLUE = ( 20, 20, 175)
33 YELLOW = (155, 155,  0)
```

```

34 LIGHTYELLOW = (175, 175, 20)
35
36 BORDERCOLOR = BLUE
37 BGCOLOR = BLACK
38 TEXTCOLOR = WHITE
39 TEXTSHADOWCOLOR = GRAY
40 COLORS  = ( BLUE, GREEN, RED, YELLOW)
41 LIGHTCOLORS = (LIGHTBLUE, LIGHTGREEN, LIGHTRED, LIGHTYELLOW)
42 assert len(COLORS) == len(LIGHTCOLORS) # each color must have light color
43
44 TEMPLATEWIDTH = 5
45 TEMPLATEHEIGHT = 5
46
47 S_SHAPE_TEMPLATE = [['.....',
48                       '.....',
49                       '..OO.',
50                       '..OO..',
51                       '.....'],
52                      ['.....',
53                       '..O..',
54                       '..OO.',
55                       '...O.',
56                       '.....']]
57
58 Z_SHAPE_TEMPLATE = [['.....',
59                       '.....',
60                       '..OO..',
61                       '..OO.',
62                       '.....'],
63                      ['.....',
64                       '..O..',
65                       '..OO..',
66                       '.O...',
67                       '.....']]
68
69 I_SHAPE_TEMPLATE = [['..O..',
70                       '..O..',
71                       '..O..',
72                       '..O..',
73                       '.....'],

```

```

74      ['.....',
75      '.....',
76      'OOOO.',
77      '.....',
78      '.....']]
79
80 O_SHAPE_TEMPLATE = [['.....',
81      '.....',
82      '.OO..',
83      '.OO..',
84      '.....']]
85
86 J_SHAPE_TEMPLATE = [['.....',
87      '.O...',
88      '.OOO.',
89      '.....',
90      '.....'],
91      ['.....',
92      '..OO.',
93      '..O..',
94      '..O..',
95      '.....'],
96      ['.....',
97      '.....',
98      '.OOO.',
99      '...O.',
100     '.....'],
101     ['.....',
102     '..O..',
103     '..O..',
104     '.OO..',
105     '.....']]
106
107 L_SHAPE_TEMPLATE = [['.....',
108     '...O.',
109     '.OOO.',
110     '.....',
111     '.....'],
112     ['.....',
113     '..O..',

```

```

114         '..O..',
115         '..OO.',
116         '.....'],
117         ['.....',
118         '.....',
119         '.OOO.',
120         '.O...',
121         '.....'],
122         ['.....',
123         '.OO..',
124         '..O..',
125         '..O..',
126         '.....']]
127
128T_SHAPE_TEMPLATE = [['.....',
129         '..O..',
130         '.OOO.',
131         '.....',
132         '.....'],
133         ['.....',
134         '..O..',
135         '..OO.',
136         '..O..',
137         '.....'],
138         ['.....',
139         '.....',
140         '.OOO.',
141         '..O..',
142         '.....'],
143         ['.....',
144         '..O..',
145         '.OO..',
146         '..O..',
147         '.....']]
148
149PIECES = {'S': S_SHAPE_TEMPLATE,
150         'Z': Z_SHAPE_TEMPLATE,
151         'J': J_SHAPE_TEMPLATE,
152         'L': L_SHAPE_TEMPLATE,
153         'I': I_SHAPE_TEMPLATE,

```

```

154     'O': O_SHAPE_TEMPLATE,
155     'T': T_SHAPE_TEMPLATE}
156
157
158def main():
159     global FPSLOCK, DISPLAYSURF, BASICFONT, BIGFONT
160     pygame.init()
161     FPSLOCK = pygame.time.Clock()
162     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
163     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
164     BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
165     pygame.display.set_caption('Tetromino')
166
167     showTextScreen('Tetromino')
168     while True: # game loop
169         if random.randint(0, 1) == 0:
170             pygame.mixer.music.load('tetrisb.mid')
171         else:
172             pygame.mixer.music.load('tetrisc.mid')
173         pygame.mixer.music.play(-1, 0.0)
174         runGame()
175         pygame.mixer.music.stop()
176         showTextScreen('Game Over')
177
178
179def runGame():
180     # setup variables for the start of the game
181     board = getBlankBoard()
182     lastMoveDownTime = time.time()
183     lastMoveSidewaysTime = time.time()
184     lastFallTime = time.time()
185     movingDown = False # note: there is no movingUp variable
186     movingLeft = False
187     movingRight = False
188     score = 0
189     level, fallFreq = calculateLevelAndFallFreq(score)
190
191     fallingPiece = getNewPiece()
192     nextPiece = getNewPiece()
193

```

```

194 while True: # game loop
195     if fallingPiece == None:
196         # No falling piece in play, so start a new piece at the top
197         fallingPiece = nextPiece
198         nextPiece = getNewPiece()
199         lastFallTime = time.time() # reset lastFallTime
200
201     if not isValidPosition(board, fallingPiece):
202         return # can't fit a new piece on the board, so game over
203
204     checkForQuit()
205     for event in pygame.event.get(): # event handling loop
206         if event.type == KEYUP:
207             if (event.key == K_p):
208                 # Pausing the game
209                 DISPLAYSURF.fill(BG_COLOR)
210                 pygame.mixer.music.stop()
211                 showTextScreen('Paused') # pause until a key press
212                 pygame.mixer.music.play(-1, 0.0)
213                 lastFallTime = time.time()
214                 lastMoveDownTime = time.time()
215                 lastMoveSidewaysTime = time.time()
216             elif (event.key == K_LEFT or event.key == K_a):
217                 movingLeft = False
218             elif (event.key == K_RIGHT or event.key == K_d):
219                 movingRight = False
220             elif (event.key == K_DOWN or event.key == K_s):
221                 movingDown = False
222
223         elif event.type == KEYDOWN:
224             # moving the piece sideways
225             if (event.key == K_LEFT or event.key == K_a) and isValidPosition(board, fallingPiece, adjX=-1):
226                 fallingPiece['x'] -= 1
227                 movingLeft = True
228                 movingRight = False
229                 lastMoveSidewaysTime = time.time()
230
231             elif (event.key == K_RIGHT or event.key == K_d) and isValidPosition(board, fallingPiece, adjX=1):
232                 fallingPiece['x'] += 1
233                 movingRight = True

```

```

234     movingLeft = False
235     lastMoveSidewaysTime = time.time()
236
237     # rotating the piece (if there is room to rotate)
238     elif (event.key == K_UP or event.key == K_w):
239         fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])
240         if not isValidPosition(board, fallingPiece):
241             fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
242     elif (event.key == K_q): # rotate the other direction
243         fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) % len(PIECES[fallingPiece['shape']])
244         if not isValidPosition(board, fallingPiece):
245             fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) % len(PIECES[fallingPiece['shape']])
246
247     # making the piece fall faster with the down key
248     elif (event.key == K_DOWN or event.key == K_s):
249         movingDown = True
250         if isValidPosition(board, fallingPiece, adjY=1):
251             fallingPiece['y'] += 1
252         lastMoveDownTime = time.time()
253
254     # move the current piece all the way down
255     elif event.key == K_SPACE:
256         movingDown = False
257         movingLeft = False
258         movingRight = False
259         for i in range(1, BOARDHEIGHT):
260             if not isValidPosition(board, fallingPiece, adjY=i):
261                 break
262             fallingPiece['y'] += i - 1
263
264     # handle moving the piece because of user input
265     if (movingLeft or movingRight) and time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ:
266         if movingLeft and isValidPosition(board, fallingPiece, adjX=-1):
267             fallingPiece['x'] -= 1
268         elif movingRight and isValidPosition(board, fallingPiece, adjX=1):
269             fallingPiece['x'] += 1
270         lastMoveSidewaysTime = time.time()
271
272     if movingDown and time.time() - lastMoveDownTime > MOVEDOWNFREQ and isValidPosition(board, fallingPiece,
adjY=1):

```



```

273     fallingPiece['y'] += 1
274     lastMoveDownTime = time.time()
275
276     # let the piece fall if it is time to fall
277     if time.time() - lastFallTime > fallFreq:
278         # see if the piece has landed
279         if not isValidPosition(board, fallingPiece, adjY=1):
280             # falling piece has landed, set it on the board
281             addToBoard(board, fallingPiece)
282             score += removeCompleteLines(board)
283             level, fallFreq = calculateLevelAndFallFreq(score)
284             fallingPiece = None
285         else:
286             # piece did not land, just move the piece down
287             fallingPiece['y'] += 1
288             lastFallTime = time.time()
289
290     # drawing everything on the screen
291     DISPLAYSURF.fill(BGCOLOR)
292     drawBoard(board)
293     drawStatus(score, level)
294     drawNextPiece(nextPiece)
295     if fallingPiece != None:
296         drawPiece(fallingPiece)
297
298     pygame.display.update()
299     FPSCLOCK.tick(FPS)
300
301
302 def makeTextObjs(text, font, color):
303     surf = font.render(text, True, color)
304     return surf, surf.get_rect()
305
306
307 def terminate():
308     pygame.quit()
309     sys.exit()
310
311
312 def checkForKeyPress():

```

```

313 # Go through event queue looking for a KEYUP event.
314 # Grab KEYDOWN events to remove them from the event queue.
315 checkForQuit()
316
317 for event in pygame.event.get([KEYDOWN, KEYUP]):
318     if event.type == KEYDOWN:
319         continue
320     return event.key
321 return None
322
323
324def showTextScreen(text):
325     # This function displays large text in the
326     # center of the screen until a key is pressed.
327     # Draw the text drop shadow
328     titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTSHADOWCOLOR)
329     titleRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))
330     DISPLAYSURF.blit(titleSurf, titleRect)
331
332     # Draw the text
333     titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTCOLOR)
334     titleRect.center = (int(WINDOWWIDTH / 2) - 3, int(WINDOWHEIGHT / 2) - 3)
335     DISPLAYSURF.blit(titleSurf, titleRect)
336
337     # Draw the additional "Press a key to play." text.
338     pressKeySurf, pressKeyRect = makeTextObjs('Press a key to play.', BASICFONT, TEXTCOLOR)
339     pressKeyRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2) + 100)
340     DISPLAYSURF.blit(pressKeySurf, pressKeyRect)
341
342     while checkForKeyPress() == None:
343         pygame.display.update()
344         FPSCLOCK.tick()
345
346
347def checkForQuit():
348     for event in pygame.event.get(QUIT): # get all the QUIT events
349         terminate() # terminate if any QUIT events are present
350     for event in pygame.event.get(KEYUP): # get all the KEYUP events
351         if event.key == K_ESCAPE:
352             terminate() # terminate if the KEYUP event was for the Esc key

```

```

353     pygame.event.post(event) # put the other KEYUP event objects back
354
355
356 def calculateLevelAndFallFreq(score):
357     # Based on the score, return the level the player is on and
358     # how many seconds pass until a falling piece falls one space.
359     level = int(score / 10) + 1
360     fallFreq = 0.27 - (level * 0.02)
361     return level, fallFreq
362
363 def getNewPiece():
364     # return a random new piece in a random rotation and color
365     shape = random.choice(list(PIECES.keys()))
366     newPiece = {'shape': shape,
367                 'rotation': random.randint(0, len(PIECES[shape]) - 1),
368                 'x': int(BOARDWIDTH / 2) - int(TEMPLATEWIDTH / 2),
369                 'y': -2, # start it above the board (i.e. less than 0)
370                 'color': random.randint(0, len(COLORS)-1)}
371     return newPiece
372
373
374 def addToBoard(board, piece):
375     # fill in the board based on piece's location, shape, and rotation
376     for x in range(TEMPLATEWIDTH):
377         for y in range(TEMPLATEHEIGHT):
378             if PIECES[piece['shape']][piece['rotation']][y][x] != BLANK:
379                 board[x + piece['x']][y + piece['y']] = piece['color']
380
381
382 def getBlankBoard():
383     # create and return a new blank board data structure
384     board = []
385     for i in range(BOARDWIDTH):
386         board.append([BLANK] * BOARDHEIGHT)
387     return board
388
389
390 def isOnBoard(x, y):
391     return x >= 0 and x < BOARDWIDTH and y < BOARDHEIGHT
392

```

```

393
394def isValidPosition(board, piece, adjX=0, adjY=0):
395     # Return True if the piece is within the board and not colliding
396     for x in range(TEMPLATEWIDTH):
397         for y in range(TEMPLATEHEIGHT):
398             isAboveBoard = y + piece['y'] + adjY < 0
399             if isAboveBoard or PIECES[piece['shape']][piece['rotation']][y][x] == BLANK:
400                 continue
401             if not isOnBoard(x + piece['x'] + adjX, y + piece['y'] + adjY):
402                 return False
403             if board[x + piece['x'] + adjX][y + piece['y'] + adjY] != BLANK:
404                 return False
405     return True
406
407def isCompleteLine(board, y):
408     # Return True if the line filled with boxes with no gaps.
409     for x in range(BOARDWIDTH):
410         if board[x][y] == BLANK:
411             return False
412     return True
413
414
415def removeCompleteLines(board):
416     # Remove any completed lines on the board, move everything above them down, and return the number of complete
417     # lines.
418     numLinesRemoved = 0
419     y = BOARDHEIGHT - 1 # start y at the bottom of the board
420     while y >= 0:
421         if isCompleteLine(board, y):
422             # Remove the line and pull boxes down by one line.
423             for pullDownY in range(y, 0, -1):
424                 for x in range(BOARDWIDTH):
425                     board[x][pullDownY] = board[x][pullDownY-1]
426             # Set very top line to blank.
427             for x in range(BOARDWIDTH):
428                 board[x][0] = BLANK
429             numLinesRemoved += 1
430             # Note on the next iteration of the loop, y is the same.
431             # This is so that if the line that was pulled down is also
432             # complete, it will be removed.
433         else:

```

```

433     y -= 1 # move on to check next row up
434     return numLinesRemoved
435
436
437 def convertToPixelCoords(boxx, boxy):
438     # Convert the given xy coordinates of the board to xy
439     # coordinates of the location on the screen.
440     return (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))
441
442
443 def drawBox(boxx, boxy, color, pixelx=None, pixely=None):
444     # draw a single box (each tetromino piece has four boxes)
445     # at xy coordinates on the board. Or, if pixelx & pixely
446     # are specified, draw to the pixel coordinates stored in
447     # pixelx & pixely (this is used for the "Next" piece).
448     if color == BLANK:
449         return
450     if pixelx == None and pixely == None:
451         pixelx, pixely = convertToPixelCoords(boxx, boxy)
452     pygame.draw.rect(DISPLAYSURF, COLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 1, BOXSIZE - 1))
453     pygame.draw.rect(DISPLAYSURF, LIGHTCOLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 4, BOXSIZE - 4))
454
455
456 def drawBoard(board):
457     # draw the border around the board
458     pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (XMARGIN - 3, TOPMARGIN - 7, (BOARDWIDTH * BOXSIZE) + 8,
459 (BOARDHEIGHT * BOXSIZE) + 8), 5)
459
460     # fill the background of the board
461     pygame.draw.rect(DISPLAYSURF, BGCOLOR, (XMARGIN, TOPMARGIN, BOXSIZE * BOARDWIDTH, BOXSIZE *
462 BOARDHEIGHT))
463     # draw the individual boxes on the board
464     for x in range(BOARDWIDTH):
465         for y in range(BOARDHEIGHT):
466             drawBox(x, y, board[x][y])
467
468 def drawStatus(score, level):
469     # draw the score text
470     scoreSurf = BASICFONT.render('Score: %s' % score, True, TEXTCOLOR)
471     scoreRect = scoreSurf.get_rect()

```

```
472 scoreRect.topleft = (WINDOWWIDTH - 150, 20)
473 DISPLAYSURF.blit(scoreSurf, scoreRect)
474
475 # draw the level text
476 levelSurf = BASICFONT.render('Level: %s' % level, True, TEXTCOLOR)
477 levelRect = levelSurf.get_rect()
478 levelRect.topleft = (WINDOWWIDTH - 150, 50)
479 DISPLAYSURF.blit(levelSurf, levelRect)
480
481
482def drawPiece(piece, pixelx=None, pixely=None):
483     shapeToDraw = PIECES[piece['shape']][piece['rotation']]
484     if pixelx == None and pixely == None:
485         # if pixelx & pixely hasn't been specified, use the location stored in the piece data structure
486         pixelx, pixely = convertToPixelCoords(piece['x'], piece['y'])
487
488     # draw each of the boxes that make up the piece
489     for x in range(TEMPLATEWIDTH):
490         for y in range(TEMPLATEHEIGHT):
491             if shapeToDraw[y][x] != BLANK:
492                 drawBox(None, None, piece['color'], pixelx + (x * BOXSIZE), pixely + (y * BOXSIZE))
493
494
495def drawNextPiece(piece):
496     # draw the "next" text
497     nextSurf = BASICFONT.render('Next:', True, TEXTCOLOR)
498     nextRect = nextSurf.get_rect()
499     nextRect.topleft = (WINDOWWIDTH - 120, 80)
500     DISPLAYSURF.blit(nextSurf, nextRect)
501     # draw the "next" piece
502     drawPiece(piece, pixelx=WINDOWWIDTH-120, pixely=100)
503
504
505if __name__ == '__main__':
506     main()
```

### Почетак кода

```
1 # Tetromino (a Tetris clone)
2 # By Al Sweigart al@inventwithpython.com
3 # http://inventwithpython.com/pygame
4 # Released under a "Simplified BSD" license
5
6 import random, time, pygame, sys
7 from pygame.locals import *
8
9 FPS = 25
10 WINDOWWIDTH = 640
11 WINDOWHEIGHT = 480
12 BOXSIZE = 20
13 BOARDWIDTH = 10
14 BOARDHEIGHT = 20
15 BLANK = '.'
```

Објаснићемо почетне константе нашег програма. Друга и трећа константа дефинишу ширину прозора наше игрице, док је квадрат широк и висок 20 пиксела. Наша табла је висока 20 квадрата и широка 10 квадрата. Константа BLANK представља празна поља на табли.

### Подешавање временских константи за притискање дугмића

```
17 MOVESIDEWAYSFREQ = 0.15
18 MOVEDOWNFREQ = 0.1
```

Сваки пут када играч притисне лево или десно дугме падајући део се помера за један квадрат улево или удесно. Међутим играч може исто тако да држи дуге лево или десно дугме како би се падајући део и даље кретао. Константа MOVESIDEWAYSFREQ ће бити подешена тако да треба да прође 0,15 секунди како би се део померио за једно место.

Константа MOVEDOWNFREQ је слична као констата MOVESIDEWAYFREQ само што се помера за једно место надоле када играч држи дугме надоле.

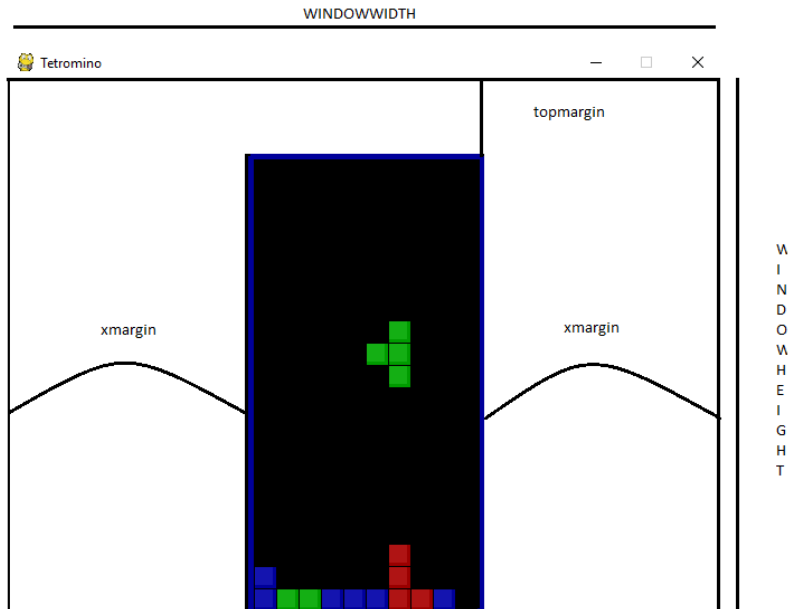
### Маргина

```
20 XMARGIN = int((WINDOWWIDTH - BOARDWIDTH * BOXSIZE) / 2)
21 TOPMARGIN = WINDOWHEIGHT - (BOARDHEIGHT * BOXSIZE) - 5
```

Програм треба да израчуна колико је празног простора између табле и прозора. Константа WINDOWWIDTH представља ширину прозора у пикселима. Наша табла има ширину константе BOARDWIDTH квадрата при чему сваки квадрат има ширину константе BOXSIZE пиксела. Ако ширину наше табле одузмемо од ширине нашег прозора добићемо празан простор између табле и прозора. Ширина прозора представља константа WINDOWWIDTH док ширину табле добијамо када BOARDWIDTH помножимо са BOXSIZE јер табла садржи BOARDWIDTH квадрата а

сваки квадрат је дужине BOXSIZE пиксела. Када поделимо са 2 добићемо празан простор са једне стране, како су и лева и десна страна исте искористићемо константу XMARGIN било за леву или десну страну.

Можемо израчунати празан простор између врха наше табле и врха прозора на сличан начин. Таблу ћемо подићи за 5 пиксела са дна прозора, па 5 одузимамо од константе TOPMARGIN.



```

23 #      R  G  B
24 WHITE   = (255, 255, 255)
25 GRAY    = (185, 185, 185)
26 BLACK    = ( 0,  0,  0)
27 RED      = (155,  0,  0)
28 LIGHTRED = (175, 20, 20)
29 GREEN    = ( 0, 155,  0)
30 LIGHTGREEN = ( 20, 175, 20)
31 BLUE     = ( 0,  0, 155)
32 LIGHTBLUE = ( 20, 20, 175)
33 YELLOW   = (155, 155,  0)
34 LIGHTYELLOW = (175, 175, 20)

```



```

35
36 BORDERCOLOR = BLUE
37 BGCOLOR = BLACK
38 TEXTCOLOR = WHITE
39 TEXTSHADOWCOLOR = GRAY
40 COLORS = ( BLUE, GREEN, RED, YELLOW)
41 LIGHTCOLORS = (LIGHTBLUE, LIGHTGREEN, LIGHTRED, LIGHTYELLOW)
42 assert len(COLORS) == len(LIGHTCOLORS) # each color must have light color

```

Постоје 4 боје у којима могу бити делови: плава, зелена, црвена и жута. Када нацртамо делове, биће за нијансу светлије. Због тога ћемо направити светлију плаву, светлију зелену, светлију црвену и светлију жуту.

Свака од ових 4 боја биће складиштена у константу COLORS (за нормалне боје) и у LIGHTCOLORS (за светлије боје).

### Подешавања шаблона за делове

```

44 TEMPLATEWIDTH = 5
45 TEMPLATEHEIGHT = 5
46
47 S_SHAPE_TEMPLATE = [['.....',
48                        '.....',
49                        '..OO.',
50                        '..OO..',
51                        '.....'],
52                       ['.....',
53                        '..O..',
54                        '..OO.',
55                        '...O.',
56                        '.....']]
57
58 Z_SHAPE_TEMPLATE = [['.....',
59                        '.....',
60                        '..OO..',
61                        '..OO.',
62                        '.....'],
63                       ['.....',
64                        '..O..',
65                        '..OO..',

```

```

66         '..O..',
67         '.....']]
68
69 I_SHAPE_TEMPLATE = [['..O..',
70         '..O..',
71         '..O..',
72         '..O..',
73         '.....'],
74         ['.....',
75         '.....',
76         'OOOO.',
77         '.....',
78         '.....']]
79
80 O_SHAPE_TEMPLATE = [['.....',
81         '.....',
82         '..OO..',
83         '..OO..',
84         '.....']]
85
86 J_SHAPE_TEMPLATE = [['.....',
87         '..O..',
88         '..OOO.',
89         '.....',
90         '.....'],
91         ['.....',
92         '..OO.',
93         '..O..',
94         '..O..',
95         '.....'],
96         ['.....',
97         '.....',
98         '..OOO.',
99         '...O.',
100        '.....'],

```

```
101      ['.....',
102      '..O..',
103      '..O..',
104      '..OO..',
105      '.....']]
106
107L_SHAPE_TEMPLATE = [['.....',
108      '..O..',
109      '...OO..',
110      '.....',
111      '.....'],
112      ['.....',
113      '..O..',
114      '..O..',
115      '..OO..',
116      '.....'],
117      ['.....',
118      '.....',
119      '...OO..',
120      '..O...',
121      '.....'],
122      ['.....',
123      '..OO..',
124      '..O..',
125      '..O..',
126      '.....']]
127
128T_SHAPE_TEMPLATE = [['.....',
129      '..O..',
130      '...OO..',
131      '.....',
132      '.....'],
133      ['.....',
134      '..O..',
135      '..OO..',
```

```

136     '..O..',
137     '.....'],
138     ['.....',
139     '.....',
140     'OOO.',
141     '..O..',
142     '.....'],
143     ['.....',
144     '..O..',
145     'OO..',
146     '..O..',
147     '.....']]

```

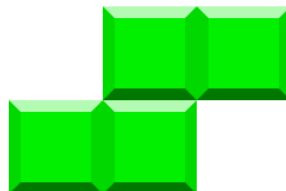
Наш програм мора да зна како настају облици наших делова, укључујући и све могуће ротације. Због тога ћемо направити листе састављене од листа које садрже стрингове. Унутрашња листа стрингова представља једну од ротацију дела, као следећа:

```

['.....',
 '.....',
 '..OO.',
 'OO..',
 '.....']

```

Цифра „0“ представља квадрате док „.“ представља празан простор.



### Раздвајање “линије кода” преко више линија

Можете да видите да се наша листа протеже кроз пар линија кода. Ово може код Python јер он док не види „]“ зна да листа још није завршена. Увлачење код Python не утиче на листу. То потврђује следећа листа:

```

spam = ['hello', 3.14, 'world', 42, 10, 'fuzz']

eggs = ['hello', 3.14,
        'world'
        , 42,
        10, 'fuzz']

```

Наравно да је тешко прочитати код под именом „eggs“ док би читљивије био део кода под именом „spam“.

Нормално, раздвајање линије кода преко више линија захтева да у нашем едитору на крају линије ставимо знак „\“. Знак „\“ каже Python да „код се наставља у следећем реду“.

Направићемо шаблон „структуре података“ облика тако што ћемо направити листу листа низова и чувати их у променљиве као што су `S_SHAPE_TEMPLATE`. На овај начин, `len(S_SHAPE_TEMPLATE)` представља колико могућих ротација постоји за облик `S` а `S_SHAPE_TEMPLATE[0]` представља прву ротацију нашег облика. Претходне слике које садрже `_SHAPE_TEMPLATE` представљају све могуће ротације за дати облик.

Замислите шаховску таблу облика 5x5 при чему су сва поља бела. Ако хоћемо да прикажемо на папир део `S` то радимо на следећи начин:

	0	1	2	3	4
0					
1					
2					
3					
4					

У коду ми записујемо `S_SHAPE_TEMPLATE[0]` као :

```
S_SHAPE_TEMPLATE[0][2][2] == 'O'
S_SHAPE_TEMPLATE[0][2][3] == 'O'
S_SHAPE_TEMPLATE[0][3][1] == 'O'
S_SHAPE_TEMPLATE[0][3][2] == 'O'
```

Наше делове везане за Тетромино у Python ћемо приказати као стрингове и листе. Константе `TEMPLATEWIDTH` и `TEMPLATEHEIGHT` представљају ширину и висину колико је велики наш део при ротацији (наш шаблон је увек 5x5).

```
149PIECES = {'S': S_SHAPE_TEMPLATE,
150          'Z': Z_SHAPE_TEMPLATE,
151          'J': J_SHAPE_TEMPLATE,
152          'L': L_SHAPE_TEMPLATE,
153          'I': I_SHAPE_TEMPLATE,
154          'O': O_SHAPE_TEMPLATE,
155          'T': T_SHAPE_TEMPLATE}
```

Променљива SHAPES је као речник која садржи све могуће шаблоне. Зато што сваки шаблон садржи све могуће ротације за дати део то значи да променљива SHAPES садржи све могуће ротације у игрици.

### Функција main ()

```
158 def main():
159     global FPSCLOCK, DISPLAYSURF, BASICFONT, BIGFONT
160     pygame.init()
161     FPSCLOCK = pygame.time.Clock()
162     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
163     BASICFONT = pygame.font.Font('freesansbold.ttf', 18)
164     BIGFONT = pygame.font.Font('freesansbold.ttf', 100)
165     pygame.display.set_caption('Tetromino')
166
167     showTextScreen('Tetromino')
```

Функција main() служи нам да направи глобалне константе и да их прикаже на почетном екрану при покретању програма.

```
168     while True: # game loop
169         if random.randint(0, 1) == 0:
170             pygame.mixer.music.load('tetrisb.mid')
171         else:
172             pygame.mixer.music.load('tetrisc.mid')
173         pygame.mixer.music.play(-1, 0.0)
174         runGame()
175         pygame.mixer.music.stop()
176         showTextScreen('Game Over')
```

Главни код се налази у функцији runGame(). Функција main() служи да насумично одабере између две музике (tetrisb.mid или tetrisc.mid) и онда позива функцију runGame() да покрене игрицу. Када играч изгуби, функција runGame() се враћа на функцију main(), која зауставља музику у позадини и на екрану исписује „Game Over“.

Када играч притисне дугме на тастатури, функција showTextScreen() која исписује „Game Over“ враћа на почетак. Игра након завршетка се увек враћа на почетак игре када изгубите.

## Почетак нове игре

```
179 def runGame():
180     # setup variables for the start of the game
181     board = getBlankBoard()
182     lastMoveDownTime = time.time()
183     lastMoveSidewaysTime = time.time()
184     lastFallTime = time.time()
185     movingDown = False # note: there is no movingUp variable
186     movingLeft = False
187     movingRight = False
188     score = 0
189     level, fallFreq = calculateLevelAndFallFreq(score)
190
191     fallingPiece = getNewPiece()
192     nextPiece = getNewPiece()
```

Пре него што игрица почне и делови крену да падају, морамо да подесимо променљиве на почетне вредности. Променљива `fallingPiece` биће постављена на дати падајући део који може бити ротиран од стране играча. Променљива `nextPiece` биће постављена као део који долази после падајућег дела тако да играч може да планира потезе унапред.

## Петља игре

```
194 while True: # game loop
195     if fallingPiece == None:
196         # No falling piece in play, so start a new piece at the top
197         fallingPiece = nextPiece
198         nextPiece = getNewPiece()
199         lastFallTime = time.time() # reset lastFallTime
200
201         if not isValidPosition(board, fallingPiece):
202             return # can't fit a new piece on the board, so game over
203
204     checkForQuit()
```

Главна петља игре управља свим кодовима главног дела игре када делови падају са врха до дна. Променљива `fallingPiece` је постављена на `None` јер је падајући део приземљен. То значи да варијабла `nextPiece` постаје `fallingPiece` што значи да следећи део постаје падајући део а `nextPiece` постаје неки насумични део. Насумични део ће произвести функција `getNewPiece()`.

Променљива `lastFallTime` је враћена на садашње време тако да време падање дела је исто секунди као и `fallFreq`.

Део који функција `getNewPiece()` насумично узима је постављен одмах изнад табле али је невидљив. Ако је ова позиција неважећа јер је табла попуњена до врха (у том случају функција `isValidPosition()` враћа вредност `false`) тада знамо да је табла попуњена и да је играч изгубио. Када се ово деси функција `runGame()` се враћа на почетак.

### Петља за руковођење догађаја

```
205     for event in pygame.event.get(): # event handling loop
206         if event.type == KEYUP:
```

Петља за руковођење догађаја обраћа пажњу кад играч ротира падајући део, помера падајући део, паузира игрицу.

### Паузирање игрице

```
207         if (event.key == K_p):
208             # Pausing the game
209             DISPLAYSURF.fill(BG_COLOR)
210             pygame.mixer.music.stop()
211             showTextScreen('Paused') # pause until a key press
212             pygame.mixer.music.play(-1, 0.0)
213             lastFallTime = time.time()
214             lastMoveDownTime = time.time()
215             lastMoveSidewaysTime = time.time()
```

Ако играч притисне дугме „p“ игра ће се паузирати. Морамо да сакријемо таблу од играча када паузира игру (у противном играч би могао да вара и стално паузира и мисли о томе где да стави падајући део).

Код празни површину са `DISPLAYSURF.fill(BG_COLOR)` и зауставља музику. Функција `showTextScreen()` исписује текст на екрану „Paused“ и чека да играч притисне дугме да би наставио.

Када притиснемо дугме враћа се екран са игрицом. `pygame.mixer.music.play(-1, 0.0)` пустиће музику испочетка у позадини. Како је прошла велика количина времена од како смо паузирали игрицу, променљиве `lastFallTime`, `lastMoveDownTime` и `lastMoveSidewaysTime` биће враћене на време пре паузе.



### Коришћење променљиве покрет за руковање корисничког уноса

```
216     elif (event.key == K_LEFT or event.key == K_a):
217         movingLeft = False
218     elif (event.key == K_RIGHT or event.key == K_d):
219         movingRight = False
220     elif (event.key == K_DOWN or event.key == K_s):
221         movingDown = False
```

Притиском на стрелице (дугмиће WASD) променљиве movingLeft, movingRight и movingDown добијају вредност False (нетачно), показујући да играч више не жели да помера падајуће делове у тим правцима. Стрелица нагоре или дугме W се користе за ротацију падајућег дела а не за померање нагоре. Због тога не постоји променљива movingUp.

### Проверавање да ли је клизање или ротирање могуће

```
223     elif event.type == KEYDOWN:
224         # moving the piece sideways
225         if (event.key == K_LEFT or event.key == K_a) and isValidPosition(board, fallingPiece,
adjX=-1):
226             fallingPiece['x'] -= 1
227             movingLeft = True
228             movingRight = False
229             lastMoveSidewaysTime = time.time()
```

Када је притиснута лева стрелица (иако је померање улево падајућег дела могуће на основу isValidPosition), тада померамо падајући део за једно место улево тако што одузмемо 1 од fallingPiece['x']. Функција isValidPosition() зависи од параметара adjX и adjY. Функција isValidPosition() проверава позицију података које пружа део објекта који се прослеђује за други параметар. Међутим, понекад не желимо да проверимо где је комад тренутно лоциран, већ неколико места изнад те позиције.

Ако прођемо за -1 adjX (кратко име за подешен X), онда се не проверава исправност позиције у структури података дела, већ коју би позицију заузео део да је једно место улево. Померање за 1 adjX означава једно место десно. Постоји и необавезни параметар adjY. Пролазећи за -1 adjY проверава једно место изнад нашег дела где је позициониран и прослеђује вредност за 3 adjY да би проверио три места надоле од нашег дела.

Променљивој movingLeft додељујемо вредност TRUE (тачно) и да се осигурамо да падајући део не би ишао и лево и десно променљивој movingRight додељујемо вредност FALSE (нетачно). Променљива lastTimeSidewaysTime биће постављена на тренутно време.

Ове променљиве су овако постављене да играч када задржи неку стрелицу део наставља да се помера. Ако је променљивој movingLeft додељена вредност True, програм зна да је играч притиснуо дугме и да није пустио дугме. Ако је прошло 0,15 секунду онда програм зна да падајући део треба да помери улево.

```

231         elif (event.key == K_RIGHT or event.key == K_d) and isValidPosition(board, fallingPiece,
adjX=1):
232             fallingPiece['x'] += 1
233             movingRight = True
234             movingLeft = False
235             lastMoveSidewaysTime = time.time()

```

Овај део кода је сличан као за део кода када притиснемо леву стрелицу (дугме A) само што важи за десну стрелицу (дугме D).

```

237         # rotating the piece (if there is room to rotate)
238         elif (event.key == K_UP or event.key == K_w):
239             fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) %
len(PIECES[fallingPiece['shape']])

```

Притиском на стрелицу горе (дугме W) падајући део се ротира у следећу ротацију. Све што код мора да уради је да повећа вредност „rotation“ за 1 у речнику fallingPiece. Али ако је повећање вредности „rotation“ веће од броја могућих ротација, онда треба прилагодити целокупном броју могућих ротација датог облика (што је len(SHAPES[fallingPiece['shape']])) и вратити га да почиње од 0.

Ево примера за прилагођавање облик J, који има 4 могуће ротације:

```

>>> 0%4
0
>>> 1%4
1
>>> 2%4
2
>>> 3%4
3
>>> 5%4
1
>>> 6%4
2
>>> 7%4
3
>>> 8%4
0
>>>

```

```

240         if not isValidPosition(board, fallingPiece):
241             fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) %
len(PIECES[fallingPiece['shape']])

```

Ако нови ротирајући облик није могућ због неких других облика на табли, тада ћемо да га вратимо у првобитан облик тако што ћемо да одузмемо 1 од fallingPiece['rotation']. Можемо да га прилагодимо и уз помоћ len(SHAPES[fallingPiece['shape']]), за вредности -1 прилагођава се тако што их враћа у првобитан облик. Ево примера за прилагођавање негативног броја:

```

>>> -1 % 4
3

```

```

242         elif (event.key == K_q): # rotate the other direction
243             fallingPiece['rotation'] = (fallingPiece['rotation'] - 1) %
len(PIECES[fallingPiece['shape']])
244         if not isValidPosition(board, fallingPiece):
245             fallingPiece['rotation'] = (fallingPiece['rotation'] + 1) %
len(PIECES[fallingPiece['shape']])

```

Овај део кода је исти као претходни само што објашњава када притиснемо дугме Q који ротира део у супротном смеру у односу на дугме W.

```

247         # making the piece fall faster with the down key
248         elif (event.key == K_DOWN or event.key == K_s):
249             movingDown = True
250             if isValidPosition(board, fallingPiece, adjY=1):
251                 fallingPiece['y'] += 1
252                 lastMoveDownTime = time.time()

```

Ако играч притисне стрелицу надоле (дугме S) онда ће падајући део падати брже него уобичајено. fallingPiece['y'] += 1 помера део за једно место надоле. Променљива movingDown је додељена вредност True и lastMoveDownTime враћа на садашње време. Ове променљиве ћемо проверити касније, тако да када држимо притиснуто дугме надоле део пада брже него уобичајено.

### Наћи доњи део

```
254     # move the current piece all the way down
255     elif event.key == K_SPACE:
256         movingDown = False
257         movingLeft = False
258         movingRight = False
259         for i in range(1, BOARDHEIGHT):
260             if not isValidPosition(board, fallingPiece, adjY=i):
261                 break
262         fallingPiece['y'] += i - 1
```

Када играч притисне размак (Space) тада дати падајући део падне одмах на дно колико може на табли. Програм прво треба да одреди колико корака мора да прође падајући део да би се приземљио.

Променљиве movingDown, movingLeft, movingRight добијају вредност False означава да играч не држи ниједно дугме. Ово је урађено јер ће код померити део до потпуног дна и почети да спушта следећи део и не желимо да изненадимо играча тако што ће делови одмах почети да падају док они држе стрелицу надоле или држе размак (Space).

Да бисмо пронашли колико најдаље може да падне наш део, прво позивамо функцију isValidPosition() и проследи цео број 1 за параметар adjY. Ако функција isValidPosition() врати вредност False онда део више не може да пада што значи да је на дну. Ако isValidPosition() врати вредност True знамо да део може да падне за још једно место наниже.

У том случају, позваћемо функцију isValidPosition() са параметром adjY чија је вредност 2. Ако врати поново вредност True онда позивамо функцију isValidPosition() са параметром adjY чија је вредност 3 и тако даље. Петља For нам служи да: позива функцију isValidPosition() да би проследила цео број параметру adjY док функција не врати вредност False. У том тренутку знамо да је вредност бројача и за 1 место испод дна. Због тога је fallingPiece['y'] једнак i-1.

BOARDHEIGHT је стављен за крајњи опсег петље јер је то максималан број поља који може да достигне део.

### Померање када задржимо стрелицу(дугме)

```
264     # handle moving the piece because of user input
265     if (movingLeft or movingRight) and time.time() - lastMoveSidewaysTime >
MOVESIDEWAYSFREQ:
266         if movingLeft and isValidPosition(board, fallingPiece, adjX=-1):
267             fallingPiece['x'] -= 1
268         elif movingRight and isValidPosition(board, fallingPiece, adjX=1):
269             fallingPiece['x'] += 1
270         lastMoveSidewaysTime = time.time()
```

Код променљиве `movingLeft` вредност је постављена `True` ако је играч притиснуо стрелицу налево(исто је и за променљиву `movingRight` када је притиснута десна стрелица). Променљиве добијају вредност `False` ако играч не држи дате стрелице(дугмиће).

Оно што се такође десило када је играч притиснуо леву или десну стрелицу је да променљива `lastMoveSidewaysTime` је постављена на тренутно време (која је била повратна вредност `time.time()`). Да је играч наставио да држи притиснуто дугме са стрелицом а да га није пустио тада би вредност променљиве `movingLeft` или `movingRight` остала `True`.

Ако држите дугме дуже од 0,15 секунди онда ће израз `time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ` бити тачан(`True`). Први `If` случај на претходној слици је тачан(`True`) ако држимо дугме(стрелицу) и прође 0,15 секунди и у том случају ће део ићи лево или десно иако не притиснемо дугме.

Ово је јако корисно за играче јер је умарајуће да кликћу дугмиће да би померили део за више места. Уместо тога могу само да задрже дугме и део ће наставити да се креће све док не скинемо прст са дугмића.

Да бисмо објаснили зашто израз `time.time() - lastMoveSidewaysTime > MOVESIDEWAYSFREQ` враћа вредност `True`(тачно), покрените следећи програм:

```
import time
WAITTIME = 4
begin = time.time()
while True:
    now = time.time()
    message = '%s, %s, %s' % (begin, now, (now - begin))
    if now - begin > WAITTIME:
        print(message + ' PASSED WAIT TIME!')
    else:
        print(message + ' Not yet...')
    time.sleep(0.2)
```

Овај програм има бесконачну петљу, да бисмо га прекинули, притисните `Ctrl+C`. Излаз овог програма ће изгледати овако:

```
1322106392.2, 1322106392.2, 0.0 Not yet...
1322106392.2, 1322106392.42, 0.219000101089 Not yet...
1322106392.2, 1322106392.65, 0.449000120163 Not yet...
1322106392.2, 1322106392.88, 0.680999994278 Not yet...
1322106392.2, 1322106393.11, 0.910000085831 Not yet...
1322106392.2, 1322106393.34, 1.1400001049 Not yet...
1322106392.2, 1322106393.57, 1.3710000515 Not yet...
1322106392.2, 1322106393.83, 1.6360001564 Not yet...
1322106392.2, 1322106394.05, 1.85199999809 Not yet...
```

```
1322106392.2, 1322106394.28, 2.08000016212 Not yet...
1322106392.2, 1322106394.51, 2.30900001526 Not yet...
1322106392.2, 1322106394.74, 2.54100012779 Not yet...
1322106392.2, 1322106394.97, 2.76999998093 Not yet...
1322106392.2, 1322106395.2, 2.99800014496 Not yet...
1322106392.2, 1322106395.42, 3.22699999809 Not yet...
1322106392.2, 1322106395.65, 3.45600008965 Not yet...
1322106392.2, 1322106395.89, 3.69200015068 Not yet...
1322106392.2, 1322106396.12, 3.92100000381 Not yet...
1322106392.2, 1322106396.35, 4.14899992943 PASSED WAIT TIME!
1322106392.2, 1322106396.58, 4.3789999485 PASSED WAIT TIME!
1322106392.2, 1322106396.81, 4.60700011253 PASSED WAIT TIME!
1322106392.2, 1322106397.04, 4.83700013161 PASSED WAIT TIME!
1322106392.2, 1322106397.26, 5.06500005722 PASSED WAIT TIME!
Traceback (most recent call last):
  File "C:\timetest.py", line 13, in
    time.sleep(0.2)
KeyboardInterrupt
```

Први број приказује вредност `time.time()` када је програм почео(ово се никад не мења). Други број је последња враћена вредност `time.time()` (ова вредност се обнавља за свако понављање петље). А трећи број је садашње време минус почетно време. Трећи број показује број секунди који је прошао откако је `begin = time.time()` линија кода извршена.

Ако је број секунди већи од 4, програм ће исписивати „PASSED WAIT TIME“ уместо „Not yet“. Овако наш програм може да зна колико је времена прошло од извршавања неке линије кода.

У нашем програму „Тетромино“, израз `time.time() - lastMoveSidewaysTime` биће процењен на број секунди када је последњи пут израз `lastMoveSidewaysTime` постављен на садашње време. Ако је ова вредност већа од вредности `MOVESIDEWAYSFREQ` онда програм зна да треба да помери падајући део за једно место.

```
272         if movingDown and time.time() - lastMoveDownTime > MOVEDOWNFREQ and
isValidPosition(board, fallingPiece, adjY=1):
273             fallingPiece['y'] += 1
274             lastMoveDownTime = time.time()
```

Овај део кода је сличан као прошли део кода само што говори о кретању падајућег дела надолу.

### Дозволити делу да „самостално“ пада

```
276     # let the piece fall if it is time to fall
277     if time.time() - lastFallTime > fallFreq:
278         # see if the piece has landed
279         if not isValidPosition(board, fallingPiece, adjY=1):
280             # falling piece has landed, set it on the board
281             addToBoard(board, fallingPiece)
282             score += removeCompleteLines(board)
283             level, fallFreq = calculateLevelAndFallFreq(score)
284             fallingPiece = None
285         else:
286             # piece did not land, just move the piece down
287             fallingPiece['y'] += 1
288             lastFallTime = time.time()
```

Брзина дела који се самостално креће доле(пада) прати променљива lastFallTime. Ако је прошло довољно времена од како се падајући део померио једно место овај део кода ће се потрудити да се падајући део помери за једно место.

Ако је израз `if not isValidPosition(board, fallingPiece, adjY=1):` тачан(`True`) онда се део приземљио. Функција `addToBoard()` нам омогућује да се будући делови могу приземљити на претходне приземљене делове док `removeCompleteLines()` служи да избрише ред који је попуњен и спусти све редове изнад за једно место. Функција `removeCompleteLines()` враћа цело број у зависности колико је редова избрисано и тај број се додаје на укупан резултат.

Пошто се резултат можда променио, позивамо функцију `calculateLevelAndFallFreq()` која служи за ажурирање тренутног нивоа и учесталости падајућег дела. За крај, променљивој `fallingPiece` додељујемо вредност `None` да би показали да следећи део постаје падајући део док следећи део треба да се насумично изабере.

Ако се део није приземљио онда вредност `Y` повећавамо за 1(помера за једно место наниже) и променљивој `lastFallTime` додељујемо садашње време.

### Цртање свих делова на екрану

```
290     # drawing everything on the screen
291     DISPLAYSURF.fill(BG_COLOR)
292     drawBoard(board)
293     drawStatus(score, level)
294     drawNextPiece(nextPiece)
295     if fallingPiece != None:
296         drawPiece(fallingPiece)
297
298     pygame.display.update()
299     FPSCLOCK.tick(FPS)
```

Сада када је петља игрице решила све догађаје и сва ажурирања, сада треба да исцрта све делове на нашем екрану. Већину цртања ће решити друге функције тако да петља треба само да их позове. Функција `pygame.display.update()` служи да се игрица појави на површини екрана а `tick()` служи да игрица не иде пребрзо.

### Функција `makeTextObjs()` која је пречица за писање текста

```
302def makeTextObjs(text, font, color):
303    surf = font.render(text, True, color)
304    return surf, surf.get_rect()
```

Функција `makeTextObjs()` нам служи као пречица. С обзиром на текст, фонт и боју она позива `render()` и враћа нам површину (`surf`) и правоугаоник(`rect`) за наш текст. Ово нам само омогућава да не куцамо сваки пут кад нам треба површина и правоугаоник.

### Функција за прекидање

```
307def terminate():
308    pygame.quit()
309    sys.exit()
```

Функција `terminate()` функционише као и у осталим програмима.



### Функција checkForKeyPress() која чека да се притисне дугме

```
312def checkForKeyPress():
313    # Go through event queue looking for a KEYUP event.
314    # Grab KEYDOWN events to remove them from the event queue.
315    checkForQuit()
316
317    for event in pygame.event.get([KEYDOWN, KEYUP]):
318        if event.type == KEYDOWN:
319            continue
320        return event.key
321    return None
```

Функција checkForKeyPress() функционише тако што позива функцију checkForQuit() која обрађује да ли има неких „QUIT“ догађаја и затвара их ако постоје. KEYDOWN представља догађај када је играч притиснуо дугме док KEYUP представља када играч склони руку са дугмета. Функција враћа вредност None само ако не постоји KEYUP догађаја.

### Функција showTextScreen() за исписивање текста на екрану

```
324def showTextScreen(text):
325    # This function displays large text in the
326    # center of the screen until a key is pressed.
327    # Draw the text drop shadow
328    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTSHADOWCOLOR)
329    titleRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2))
330    DISPLAYSURF.blit(titleSurf, titleRect)
331
332    # Draw the text
333    titleSurf, titleRect = makeTextObjs(text, BIGFONT, TEXTCOLOR)
334    titleRect.center = (int(WINDOWWIDTH / 2) - 3, int(WINDOWHEIGHT / 2) - 3)
335    DISPLAYSURF.blit(titleSurf, titleRect)
336
337    # Draw the additional "Press a key to play." text.
338    pressKeySurf, pressKeyRect = makeTextObjs('Press a key to play.', BASICFONT, TEXTCOLOR)
339    pressKeyRect.center = (int(WINDOWWIDTH / 2), int(WINDOWHEIGHT / 2) + 100)
340    DISPLAYSURF.blit(pressKeySurf, pressKeyRect)
```

Уместо да направимо две различите функције за почетни екран и за крај игре, направићемо једну општу функцију под називом `showTextScreen()`. Функција `showTextScreen()` ће написати текст који ми напишемо. Функција ће исписати на екран „Press a key to play“ (притисни дугме да би играо).

Први пасус исписује текст тамне боје, док други пасус исписује исти тај текст само 3 пиксела улево и 3 пиксела ка горе. Овако добијамо „Drop shadow“ ефекат који даје бољи изглед тексту. Можете да видите разлику тако што први пасус ставите као коментар. Функција `showTextScreen()` нам служи за почетну слику игрице, за крајњу слику игрице и за паузиран део игрице (који ћемо објаснити касније).

```
342 while checkForKeyPress() == None:
343     pygame.display.update()
344     FPSLOCK.tick()
```

Желимо да текст стоји на екрану док играч не притисне дугме. Ова мала петља ће непрекидно позивати функције `pygame.display.update()` и `FPSLOCK.tick()` док функција `checkForKeyPress()` не врати вредност која није `None`. Ово се деси када играч притисне дугме.

### Функција `checkForQuit()`

```
347 def checkForQuit():
348     for event in pygame.event.get(QUIT): # get all the QUIT events
349         terminate() # terminate if any QUIT events are present
350     for event in pygame.event.get(KEYUP): # get all the KEYUP events
351         if event.key == K_ESCAPE:
352             terminate() # terminate if the KEYUP event was for the Esc key
353     pygame.event.post(event) # put the other KEYUP event objects back
```

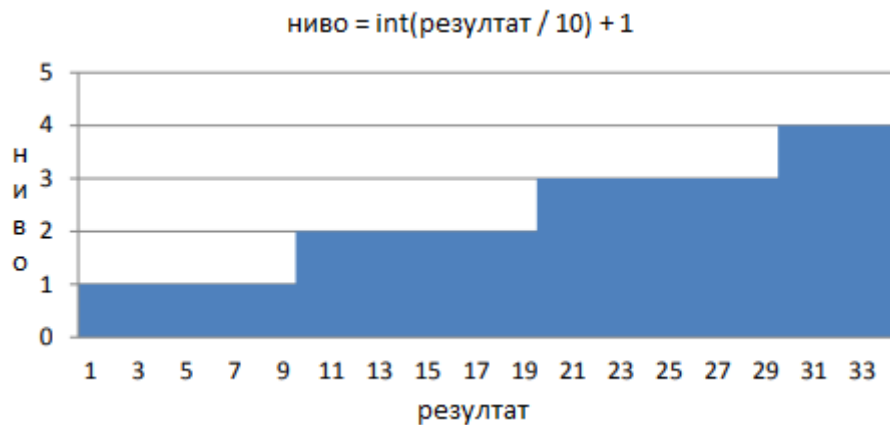
Функција `checkForQuit()` решава било који догађај који хоће да обустави рад програма. Ово се дешава ако постоји „QUIT“ догађаја који су у реду догађаја или ако је неко подигао прст са Esc дугме. Играч треба бити у могућности да било кад притисне Esc дугме да прекине игру.

### Функција `calculateLevelAndFallFreq()`

```
356 def calculateLevelAndFallFreq(score):
357     # Based on the score, return the level the player is on and
358     # how many seconds pass until a falling piece falls one space.
359     level = int(score / 10) + 1
360     fallFreq = 0.27 - (level * 0.02)
361     return level, fallFreq
```

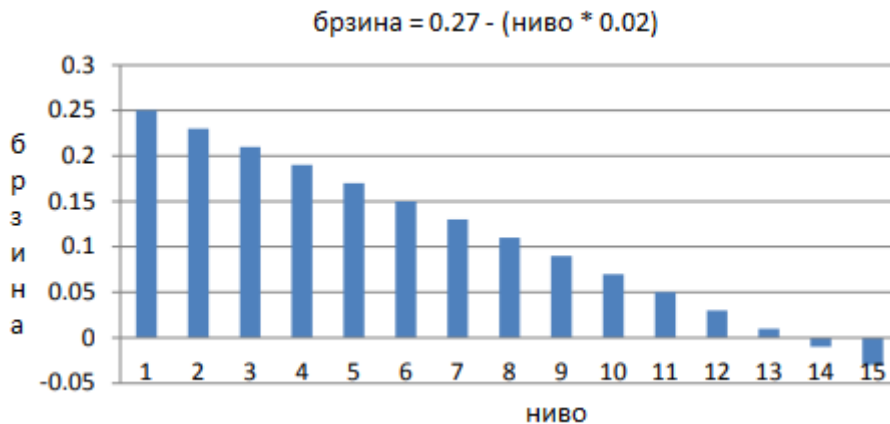
Сваки пут када играч попуни линију добија један поен. Након сакупљених 10 поена играч прелази на виши ниво и падајући делови постају бржи. Ова функција нам служи за израчунавање нивоа и фреквенције падајућег дела за сваки пређени ниво.

Променљива level(ниво) је инт јер сваки број подељен са 10 који има остатак биће заокругљен на мању цифру. Ако је променљива score између 0 и 9 онда ће овај израз бити 0 и плус 1 даје први level(ниво) све док променљива score не пређе 10. Да бисте схватили погледајте следећи график:

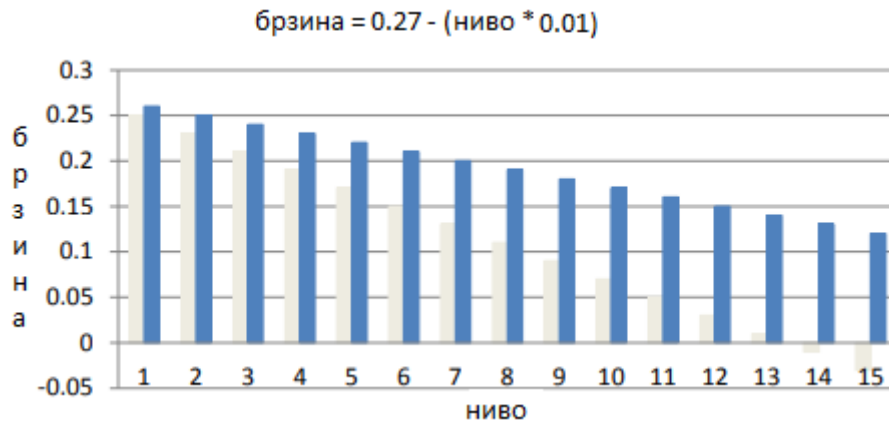


Да бисте израчунали брзину падања дела, узећемо пример да је брзина 0,27 секунди(сваких 0,27 део се помери за једно место на доле). Онда ниво(level) помножимо са 0,02 и одузмемо од 0,27. Ако је ниво први онда брзину добијамо  $0,27 - (1 * 0,02)$ , за други ниво  $0,27 - (2 * 0,02)$  и тако даље. За сваки ниво део ће падати за 0,02 секунде брже.

График који приказује брзину за сваки ниво:



Видимо да је брзина за 14. ниво негативна, ако је FPS постављен на 25 то значи да ће наш падајући део за једну секунду пасти 25 места а наша табла има 20 места у висину. Да бисте избегли ово можете променљиву fallFreq изразити овако  $0,27 - (\text{score} * 0,01)$  тј. да се брзина повећава дупло мање него прошли пут.



Видимо да би у овом случају 14. ниво био тежак као првобитни 7. ниво. Игрицу мењамо да буде тежа или лакша мењањем једначина у функцији `calculateLevelAndFallFreq()`.

### Функција `getNewPiece()` за прављење нових делова

```

363 def getNewPiece():
364     # return a random new piece in a random rotation and color
365     shape = random.choice(list(PIECES.keys()))
366     newPiece = {'shape': shape,
367                 'rotation': random.randint(0, len(PIECES[shape]) - 1),
368                 'x': int(BOARDWIDTH / 2) - int(TEMPLATEWIDTH / 2),
369                 'y': -2, # start it above the board (i.e. less than 0)
370                 'color': random.randint(0, len(COLORS)-1)}
371     return newPiece

```

Функција `getNewPiece()` прави насумичне делове које позиционира на врху табле. Прво насумично бирамо облик дела тако што позовемо листу са свим могућим облицима. Функција `random.choice()` бира насумично један облик. Тај облик постаје нови део, при чему је нови део(`newPiece`) у облику речника. „rotation“ показује у ком облику ротације ће бити наш део. „x“ нам служи да наш део буде постављен на средини док „y“ нам каже да ће бити на врху табле. „color“ нам даје насумичну боју нашем падајућем делу.

### Додавање дела на таблу

```

374 def addToBoard(board, piece):
375     # fill in the board based on piece's location, shape, and rotation
376     for x in range(TEMPLATEWIDTH):
377         for y in range(TEMPLATEHEIGHT):
378             if PIECES[piece['shape']][piece['rotation']][y][x] != BLANK:
379                 board[x + piece['x']][y + piece['y']] = piece['color']

```

Структура података табле представља приказ података на простору облика правоугаоника где се налазе делови који су претходно приземљени. Део који тренутно пада није означен на табли. Функција `addToBoard()` узима податке и поставља их на таблу када се приземљи (обележи квадрате које заузима део).

For петља пролази кроз сваки празан простор структуре података дела и ако нађе поље у простору додаје га на табли.

### Креирање нове структуре података табле

```
382def getBlankBoard():
383    # create and return a new blank board data structure
384    board = []
385    for i in range(BOARDWIDTH):
386        board.append([BLANK] * BOARDHEIGHT)
387    return board
```

Структура података која се користи за таблу је једноставна: то је листа листи са вредностима. Ако је вредност једнака вредности `BLANK` онда је то празно поље. Ако је вредност цео број онда то представља квадрат. Број 0 је плава боја, број 1 зелена, број 2 је црвена, броје 4 је жута.

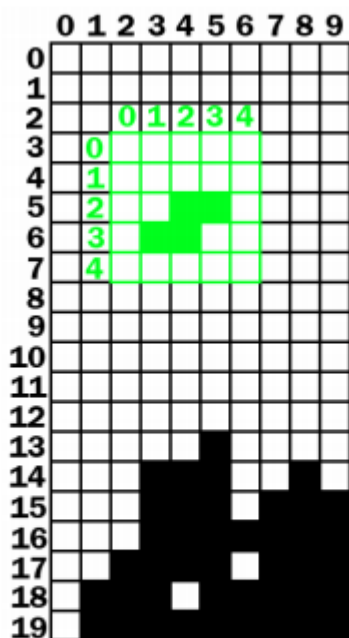
### Функције `isOnBoard()` и `isValidPosition()`

```
390def isOnBoard(x, y):
391    return x >= 0 and x < BOARDWIDTH and y < BOARDHEIGHT
```

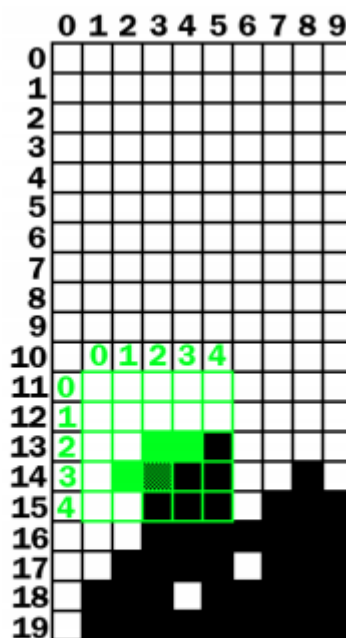
Функција `isOnBoard()` служи да проверава да ли дата координата одговара табли. Да ли је ху координата у распону од 0 до висине табле и ако јесте враћа `True`(тачно).

```
394def isValidPosition(board, piece, adjX=0, adjY=0):
395    # Return True if the piece is within the board and not colliding
396    for x in range(TEMPLATEWIDTH):
397        for y in range(TEMPLATEHEIGHT):
398            isAboveBoard = y + piece['y'] + adjY < 0
399            if isAboveBoard or PIECES[piece['shape']][piece['rotation']][y][x] == BLANK:
400                continue
401            if not isOnBoard(x + piece['x'] + adjX, y + piece['y'] + adjY):
402                return False
403            if board[x + piece['x'] + adjX][y + piece['y'] + adjY] != BLANK:
404                return False
405    return True
```

Функцији isValidPosition() је дата структура података табле и структура података дела, и враћа вредност True ако сви квадрати у делу на табли и не поклапају се са било којим квадратом на табли. Ово се ради тако што се узимају XY координате (који је представљен помоћу квадрата 5x5) и додавањем координате унутар структуре података дела. Ево пар примера који показују:



Табла са падајућим  
делом у важећем  
положају



Табла са падајућим  
делом у неважећем  
положају

На левој табли део који пада (горњи леви угао) има координате (2,3) на табли. Али квадрати унутар координатног система падајућег дела имају своје координате. Да би ови делови нашли координате „табле“, само треба да додамо координате „табле“ левог падајућег дела и део координате квадрата.

На левој табли, квадрати падајућег дела имају координате:

(2,2), (3,2), (1,3), (2,3)

Када додамо координате (2,3) овим координатама добијамо:

(2+2,2+3), (3+2,2+3), (1+2,3+3), (2+2,3+3)

Након додавања координате (2,3) квадрати имају дате координате:

(4,5), (5,5), (3,6), (4,6)

Сада можемо да откријемо квадрате падајућих делова координата табле, можемо да видимо да ли се оне поклапају са приземљеним деловима на табли. Угњеждена петља пролази кроз сваку могућу координату која је на приземљеном делу.

Желимо да проверимо да ли је падајући део искључен са табле или се поклапа са делом на табли. Променљива isAboveBoard има вредност True ако оквир падајућег дела има координате x која се поклапа и у тако да је изнад табле. У супротном је False.

Исказ IF проверава да ли је простор дела изнад табле или је празан. Ако је било која вредност тачна(True) и програм наставља са радом.

Други IF исказ проверава да ли је оквир падајућег дела лоциран на табли. Трећи IF проверава да ли није унутар простора табле лоциран празан оквир падајућег дела. Ако је неки од ових исказа тачан наша функција враћа вредност True.

Ако код прође кроз угњеждену петљу FOR и не враћа вредности False онда је позиција дела важећа и функција враћа вредност True.

#### Провера и брисање реда ако је попуњен

```
407 def isCompleteLine(board, y):
408     # Return True if the line filled with boxes with no gaps.
409     for x in range(BOARDWIDTH):
410         if board[x][y] == BLANK:
411             return False
412     return True
```

Функција isCompleteLine провера ред на основу у координате. Ред кажемо да је попуњен ако нема празног квадрата. Петља FOR пролази кроз сваки квадрат у реду. Ако постоји квадрат у реду који је празан онда функција враћа False вредност.

```
415 def removeCompleteLines(board):
416     # Remove any completed lines on the board, move everything above them down, and return
the number of complete lines.
417     numLinesRemoved = 0
418     y = BOARDHEIGHT - 1 # start y at the bottom of the board
419     while y >= 0:
```

Функција removeCompleteLines() тражи попуњени ред, брише га и спушта све квадрате за једно место који су изнад тог реда. Функција ће вратити вредност која представља колико је реда уклоњено и додаће ту вредност на резултат(Score).

Ово функционише због петље While која провера координату у која креће од BOARDHEIGHT-1 (одузимамо 1 због врха) и ради све док у не постане негативан цео број.

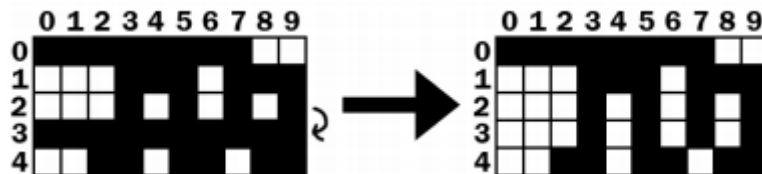
```

420 if isCompleteLine(board, y):
421     # Remove the line and pull boxes down by one line.
422     for pullDownY in range(y, 0, -1):
423         for x in range(BOARDWIDTH):
424             board[x][pullDownY] = board[x][pullDownY-1]
425     # Set very top line to blank.
426     for x in range(BOARDWIDTH):
427         board[x][0] = BLANK
428     numLinesRemoved += 1
429     # Note on the next iteration of the loop, y is the same.
430     # This is so that if the line that was pulled down is also
431     # complete, it will be removed.
432 else:
433     y -= 1 # move on to check next row up
434 return numLinesRemoved

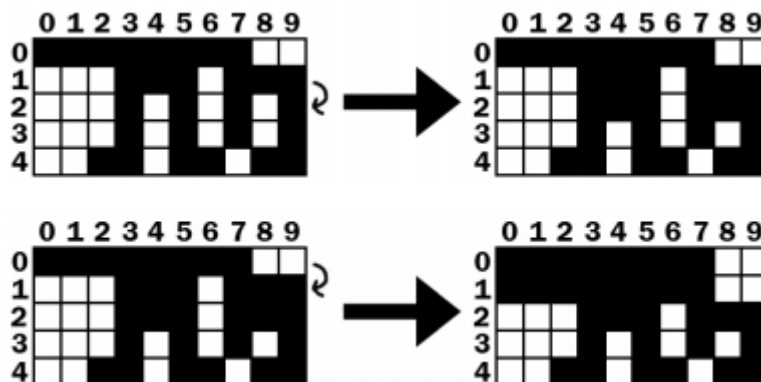
```

Функција `isCompleteLine()` враћа вредност `True` за дату вредност `y` ако је ред попуњен. У случају да програм треба да копира вредности за сваки ред који је изнад обрисаног реда.

Погледајмо следећи пример. Да бисмо сачували простор показаћемо само првих пет редова са врха. Ред 3 је попуњен па се брише и повлачи све квадрате који су изнад за једно место. Табла десно показује како ће изгледати након померања:

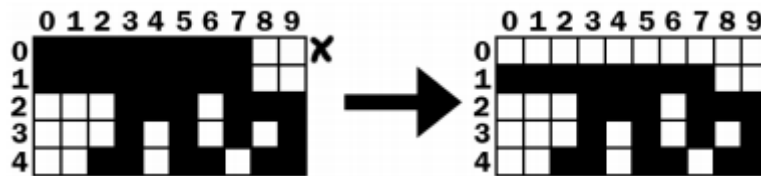


Повлачење надоле представља копирање вредности редова изнад и пребацује их ка доле. Након копирања ред 2 на ред 3, ред 1 се пребацује на ред 2 и ред 0 се пребацује на ред 1:



Ред 0 нема ред изнад да копира на себе. Ред 0 и не треба то, довољно је да цео ред буде празан. Након тога табла ће направити промену као што је на слици испод:





### Пребацивање са координата табле на координате пиксел

```

437 def convertToPixelCoords(boxx, boxy):
438     # Convert the given xy coordinates of the board to xy
439     # coordinates of the location on the screen.
440     return (XMARGIN + (boxx * BOXSIZE)), (TOPMARGIN + (boxy * BOXSIZE))

```

Ова функција нам омогућава да пратимо координате табле на координате пиксел. Исто функционише као и пребацивање у претходним поглављима.

### Цртање квадрата на табли и негде на екрану

```

443 def drawBox(boxx, boxy, color, pixelx=None, pixely=None):
444     # draw a single box (each tetromino piece has four boxes)
445     # at xy coordinates on the board. Or, if pixelx & pixely
446     # are specified, draw to the pixel coordinates stored in
447     # pixelx & pixely (this is used for the "Next" piece).
448     if color == BLANK:
449         return
450     if pixelx == None and pixely == None:
451         pixelx, pixely = convertToPixelCoords(boxx, boxy)
452     pygame.draw.rect(DISPLAYSURF, COLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 1, BOXSIZE - 1))
453     pygame.draw.rect(DISPLAYSURF, LIGHTCOLORS[color], (pixelx + 1, pixely + 1, BOXSIZE - 4, BOXSIZE - 4))

```

Функција `drawBox()` служи за цртање квадрата на табли. Функцији дајемо параметре `boxx` и `boxy` координате табле где треба бити нацртан квадрат. Међутим ако су параметри `pixelx` и `pixely` наведене (одређене) онда ће оне надјачати параметре `boxx` и `boxy`. `pixelx` и `pixely` служе за цртање оквира следећег дела који није на табли.

Ако параметри `pixelx` и `pixely` нису постављени тада ће њихова вредност бити `None` јер тако ради функција кад се покрене. Функција `convertToPixelCoords()` ће одредити вредности `pixelx` и `pixely` на онсову вредности `boxx` и `boxy`.

Код неће попунити цели простор бојом. `pygame.draw.rect()` нам омогућава да имамо линије између квадрата.

### Цртање по екрану

```
456def drawBoard(board):
457     # draw the border around the board
458     pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (XMARGIN - 3, TOPMARGIN - 7,
459 (BOARDWIDTH * BOXSIZE) + 8, (BOARDHEIGHT * BOXSIZE) + 8), 5)
459
460     # fill the background of the board
461     pygame.draw.rect(DISPLAYSURF, BGCOLOR, (XMARGIN, TOPMARGIN, BOXSIZE *
462 BOARDWIDTH, BOXSIZE * BOARDHEIGHT))
462     # draw the individual boxes on the board
463     for x in range(BOARDWIDTH):
464         for y in range(BOARDHEIGHT):
465             drawBox(x, y, board[x][y])
```

Функција `drawBoard()` служи за цртање ивице табле и квадрата унутар табле. Прво се црта ивица табле на `DISPLAYSURF`, па се боји позадина табле. Функција `drawBox()` нам служи за цртање квадрата унутар табле.

### Писање резултата и нивоа

```
468def drawStatus(score, level):
469     # draw the score text
470     scoreSurf = BASICFONT.render('Score: %s' % score, True, TEXTCOLOR)
471     scoreRect = scoreSurf.get_rect()
472     scoreRect.topleft = (WINDOWWIDTH - 150, 20)
473     DISPLAYSURF.blit(scoreSurf, scoreRect)
474
475     # draw the level text
476     levelSurf = BASICFONT.render('Level: %s' % level, True, TEXTCOLOR)
477     levelRect = levelSurf.get_rect()
478     levelRect.topleft = (WINDOWWIDTH - 150, 50)
479     DISPLAYSURF.blit(levelSurf, levelRect)
```

Функција `drawStatus()` служи за представљање резултата и нивоа у горњем десном углу.

### Цртање делова по екрану

```
482 def drawPiece(piece, pixelx=None, pixely=None):
483     shapeToDraw = PIECES[piece['shape']][piece['rotation']]
484     if pixelx == None and pixely == None:
485         # if pixelx & pixely hasn't been specified, use the location stored in the piece data structure
486         pixelx, pixely = convertToPixelCoords(piece['x'], piece['y'])
487
488     # draw each of the boxes that make up the piece
489     for x in range(TEMPLATEWIDTH):
490         for y in range(TEMPLATEHEIGHT):
491             if shapeToDraw[y][x] != BLANK:
492                 drawBox(None, None, piece['color'], pixelx + (x * BOXSIZE), pixely + (y * BOXSIZE))
```

Функција `drawPiece()` црта оквире делова према структури података која им је прослеђена. Ова функција нам служи за цртање падајућег дела и за цртање следећег падајућег дела. Пошто структура података о делу садржи све информације о облику, позиције, ротације и боје, ништа друго осим структуре података о делу не треба да проследи функцији.

Међутим следећи део није нацртан на табли. У овом случају, игноришемо информације о положају унутар структуре података о делу и уместо тога функција `drawPiece()` прослеђује аргументе за параметре `pixelx` и `pixely` како би одредили где тачно да се нацртају делови.

Петље FOR нам служе за цртање квадратића на нашим деловима које треба нацртати.

### Цртање следећег дела

```
495 def drawNextPiece(piece):
496     # draw the "next" text
497     nextSurf = BASICFONT.render('Next:', True, TEXTCOLOR)
498     nextRect = nextSurf.get_rect()
499     nextRect.topleft = (WINDOWWIDTH - 120, 80)
500     DISPLAYSURF.blit(nextSurf, nextRect)
501     # draw the "next" piece
502     drawPiece(piece, pixelx=WINDOWWIDTH-120, pixely=100)
503
504
505 if __name__ == '__main__':
506     main()
```

Функција `drawNextPiece()` црта следећи део који се налази у горњем десном углу. Цртамо са функцијом `drawPiece()` којој прослеђујемо аргументе пиџелx и пиџелy.

Ово је последња функција. Последњи пасус нам служи да након што су све функције извшене онда функција `main()` покреће главни програм игрице.

## Резиме

Тетромино је игрица у којој блокови падају са врха табле и играч их помера и ротира тако да попуни ред. Ред нестаје и све блокови падају који су изнад. Игра се завршава када блокови попуне средину до врха.

Оригиналну игрицу направио је човек по имену „Алекс Пајитнов“ из Совјетског савеза 1984 године. Игрица је забавна, једноставна и заразна. Игрица је једна од најпопуларнијих игрица икада продана у 100 милиона примерака и има доста својих верзија.

И све то је направила једна особа која зна да програмира.

Ако имате идеја и неко основно знање програмирања можете да направите невероватно забавну игрицу. Са мало вежбе можете ваше идеје за игрице да претворите у праве програме који могу постати познати као и „Тетрис“.

За вежбање програмирања идите на линк <http://invpy.com/buggy/tetromino>

Постоји доста варијација игрице Тетромино. Неке од њих су „Pentamino“ где су модели састављени од 5 квадрата. Постоји и „Tetromino for idiots“ где су сви модели направљени од једног квадрата.

Ове варијације можете скинути на линкове:

- <http://invpy.com/pentomino.py>
- <http://invpy.com/tetrominoforidiots.py>