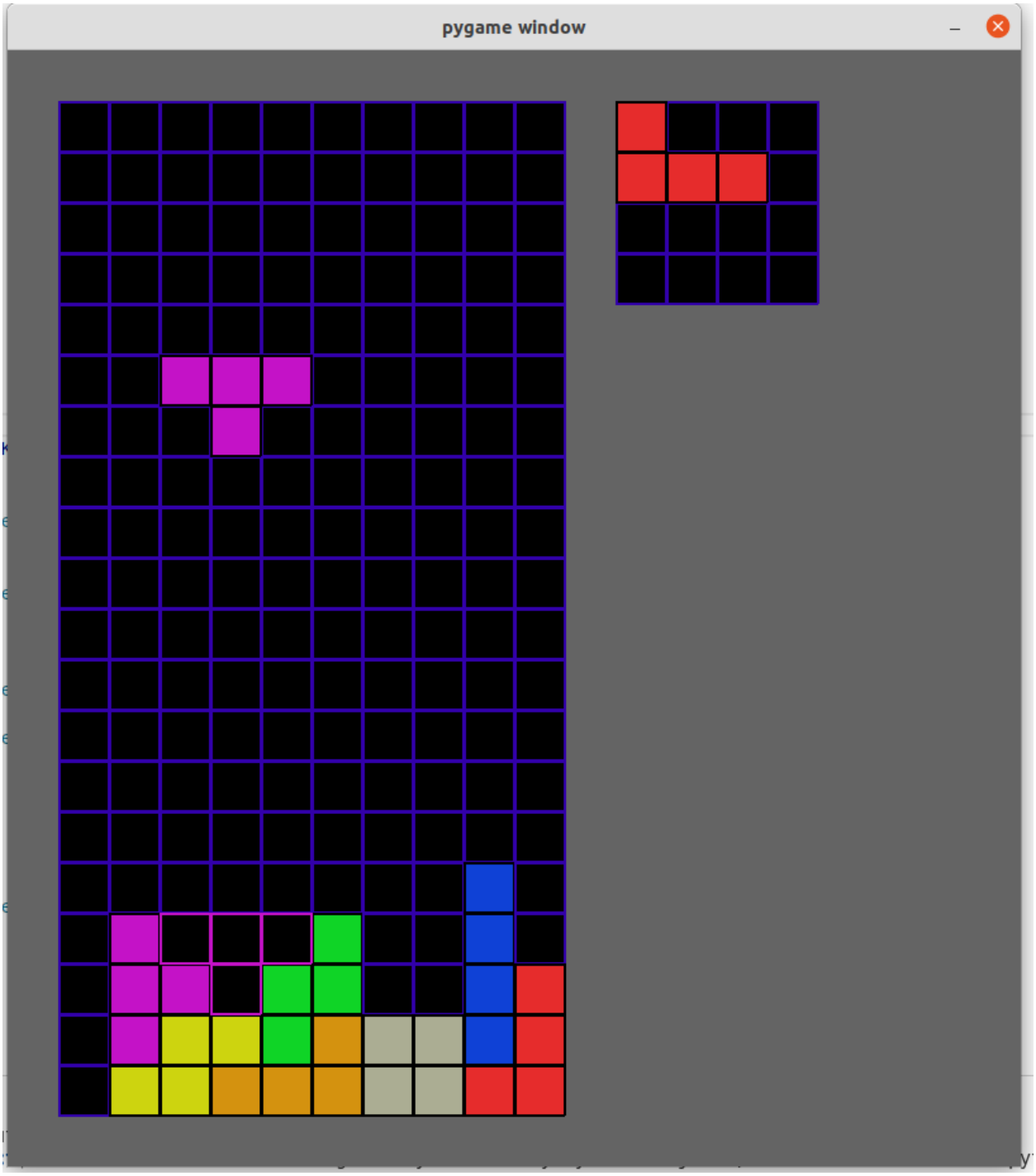
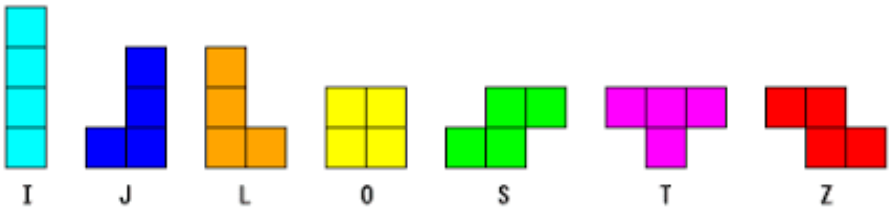
Tetris c Pygame



Речник:

Tetromino - Наименование на фигурите образувани от квадратчета. Те са общо 7 наименувани според приликата им с латинска буква:
I, J, L, O, S, T, Z



460 × 120

Съдържание:

Game mechanics/Игрови механики

1. Избор на tetromino което да се появи

Използва се координатна система с Y надолу и X надясно
Всяко тетромино съдържа лист от кортежи, всеки един с координати на съставен квадрат на тетрамината

tetromino.py

```
spawns = [
    [(1, 1), (2, 1), (0, 1), (3, 1)],    # I
    [(1, 1), (0, 1), (1, 0), (2, 1)],    # T
    [(1, 1), (0, 1), (0, 0), (2, 1)],    # J
    [(1, 1), (0, 1), (2, 1), (2, 0)],    # L
    [(1, 1), (0, 1), (1, 0), (2, 0)],    # Z
    [(1, 1), (1, 0), (0, 0), (2, 1)],    # S
```

```
    [(1, 1), (2, 0), (1, 0), (2, 1)],    # O
]
```

Изборът става по следния начин. Всяко едно от видовете тетроминота трябва да се появи веднъж , но в случаен ред, след което, отново по веднъж с нов случаен ред

```
# current tetromino set until new are generated
tetromino_set = []
```

Съществува лист който съдържа тетроминота по реда на появяване.

При появяване на тетромино се взима първото от листа и се премахва от него.

```
def get_tetromino():
    t = tetromino_set.pop(0)
    if len(tetromino_set) == 0:
        gen_tetromino_set()

    return t
```

Ако листа се изчерпи се генерира нов случаен ред

```
def gen_tetromino_set():
    type_list = list(range(0, 7))
    random.shuffle(type_list)

    for type in type_list:
        t = Tetromino(type)
        t.move_no_coll(3, -2)
        tetromino_set.insert(0,t)
```

Първо се генерира лист с числата от 0 до 6 с случаен ред
Тези числа предадени към конструктура на Tetromino се използват като индекс на `spawns` и така Tetromino се сдобива с self.tiles, листа с координатите на индивидуалните квадратчета

```
class Tetromino:
    def __init__(self, type):
        # print(spawns)
        self.color = colors[type]
        self.tiles = spawns[type].copy()
        self.is_ghost = False
        self.type = type
        self.rotation_state = 0
```

В main се провокира генерирането на първия лист от тетроминота и се взима първия още преди main цикъла

```
tetromino.gen_tetromino_set()
t = tetromino.get_tetromino()
```

2. Движение на тетромино

2.1. Наляво-надясно

Има два вида движение:

2.1.1. Просто - при единично натискане на бутон ляво/дясно

```
main.py #main loop
while running:
...
elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_d or event.key == pygame.K_RIGHT:
                right_pressed = True
                t.move(1, 0)
            elif event.key == pygame.K_a or event.key == pygame.K_LEFT:
                left_ressed = True
                t.move(-1, 0)
```

То става чрез викане еднократно метода move на Tetromino при натискане на бутон

2.1.2. При задържане на бутон ляво/дясно

Тук нещата са по сложни.

Трябва, при натискане и до отпускане на клавиша, през даден интервал от милисекунди, да се осъществи движение чрез викане на move. Затова са необходими булевите променливи left_pressed и right_pressed.

```
elif event.type == pygame.KEYUP:
            if event.key == pygame.K_d or event.key == pygame.K_RIGHT:
                right_pressed = False
            elif event.key == pygame.K_a or event.key == pygame.K_LEFT:
                left_ressed = False
```

Интервала се реализира по надолу в main-loop-а така:

```
            if curr_ticks - x_move_ticks >= HORIZONTAL_SPEED:
                t.move(x_move, 0)
                x_move_ticks = curr_ticks
```

curr_ticks са милисекундите от началото на програмта до момента

x_move_ticks са милисекундите от началото на програмата когато последно е извършено движение наляво или надясно

Израза `curr_ticks - x_move_ticks` ще бъде колко време е минало от последното автоматично движение наляво/надясно. Следователно целият блок код отгоре, ако бива повикван всеки frame(кадър), ще извърша хоризонтално движение на всеки:
`HORIZONTAL_SPEED = 70`
70 милисекунди в нашичт случай.

```
   # Horizontal movement
while running:
...
   if not debug_mode:
       if left_ressed != right_pressed:
           if left_ressed:
               x_move = -1
           elif right_pressed:
               x_move = 1
       else:
           x_move = 0

       if curr_ticks - das_delay_ticks >= DAS_DELAY:
           if curr_ticks - x_move_ticks >= HORIZONTAL_SPEED:
               t.move(x_move, 0)
               x_move_ticks = curr_ticks
```

Цялата логика за продължителното хоризонтално движение изглежда така. Игнорирайте първия ред с debug_mode засега.
В началото се проверява да не би двата бутона наляво и надясно не са натиснати едновременно, ако е само един, определяме посока на движение, ако са два, няма движение.
Има и нов ред който не съм обеснил досега и той е:
`if curr_ticks - das_delay_ticks >= DAS_DELAY:`

По логика е аналогичен с долният и се добавя с цел автоматичното хоризонтално движение като процес да започне само след по продължително задържане, за да се избегне човешката грешка кадето някой желае само еднократно движение, но без да иска не е отпуснал бутона достатъчно бързо. От лично тестване така е много по- удобно и по-лесно, като играч, да реализирам движенията които желая.
https://tetris.fandom.com/wiki/DAS

```
DAS_DELAY = 250
```

2.2. Надолу

Движението надолу се извършва автоматично през даден интервал от време, при задържане на бутон надолу, този интервал става по-къс, съответно движението по-бързо.

```
elif event.type == pygame.KEYDOWN:
...
            elif event.key == pygame.K_s or event.key == pygame.K_DOWN:
                down_pressed = True

 elif event.type == pygame.KEYUP:
...
            elif event.key == pygame.K_s or event.key == pygame.K_DOWN:
                down_pressed = False
```

Аналогично като предходно разглежданото автоматично движение (наляво-надясно) използваме булева променлива за следене дали бутона е натиснат и задържан

```
   # Vertical movement
   #Ignore if not debug_mode: #Ignore
       curr_ticks = pygame.time.get_ticks()
       if (down_pressed and curr_ticks - y_move_ticks >= FAST_DOWN_SPEED)\
           or (curr_ticks - y_move_ticks >= NORMAL_DOWN_SPEED):
               t.move(0, 1)
               y_move_ticks = curr_ticks
```

Впоследвие метода move бива викан, ако е натиснат и задържан бутона, през един интервал от време, а ако не, с друг.

```
NORMAL_DOWN_SPEED = 800
FAST_DOWN_SPEED = 50
```

Нека сега разгледаме самия move метод:

```
class Tetromino:
...
   def move(self, x, y):
       prev_tiles = self.tiles.copy()
       self.move_no_coll(x, y)

       if self.check_collision():
           self.tiles = prev_tiles
```

```
            # raised reached floor event
            if (y >= 1):
                pygame.event.post(pygame.event.Event(REACHED_FLOOR))

        sound.play_sound("movement")
```

x - желано хоризонтално движение
y - желано вертикално движение

Prev_tiles - първо взимаме копие на списъка с координатите на квадратчета съставящи тетроминото преди каквито и да е движения(промени) на тези координати да се извършат. Тази променлива е всъщност оригиналната позиция на тетроминото, тя ще ни служи след малко.

self.move_no_coll(x, y) извъшва елементарна операция над self.tiles

```
    def move_no_coll(self, x, y):
         for i in range(0, len(self.tiles)):
            self.tiles[i] = (self.tiles[i][0] + x, self.tiles[i][1] + y)
```

Променя x(индекс 0 в tuple-а) и y(индекс 1) стойността на всяко квадратче с дадените аргументи x, y. Реално този метод извършва самото движение.

```
        if self.check_collision():
            self.tiles = prev_tiles
            # raised reached floor event
            if (y >= 1):
                pygame.event.post(pygame.event.Event(REACHED_FLOOR))

        sound.play_sound("movement")
```

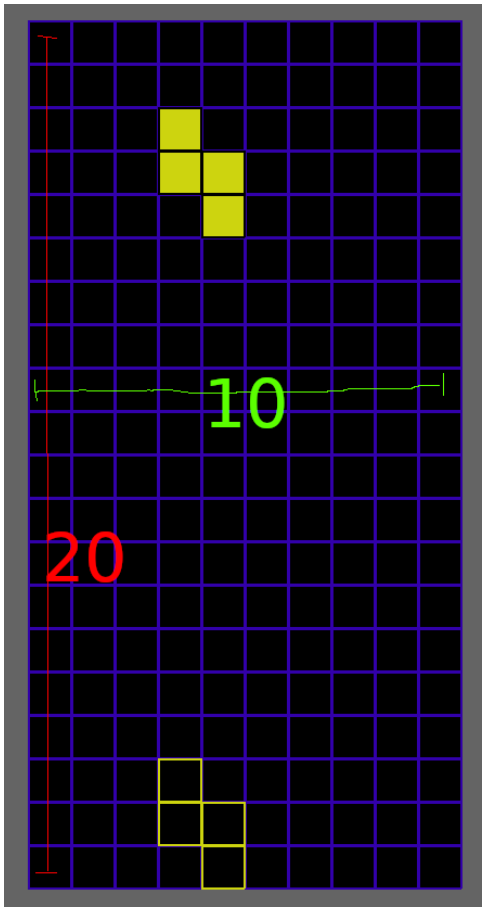
След това се връщаме в move метода където се вика self.check_collision() метода.

```
def check_collision(self):
        for tile in self.tiles:
            # Reach floor
            if tile[1] > world.GAME_HEIGHT - 1:
                return True
            # Reach side walls
            if tile[0] < 0 or world.GAME_WIDTH - 1 < tile[0]:
                return True

            # Reach locked tetrominos solid grid
            if (tile[0] >= 0 and tile[1] >= 0):
                if world.grid[tile[1]][tile[0]] != world.GRID_COLOR_FILL:
                    return True
```

```
        return False
```

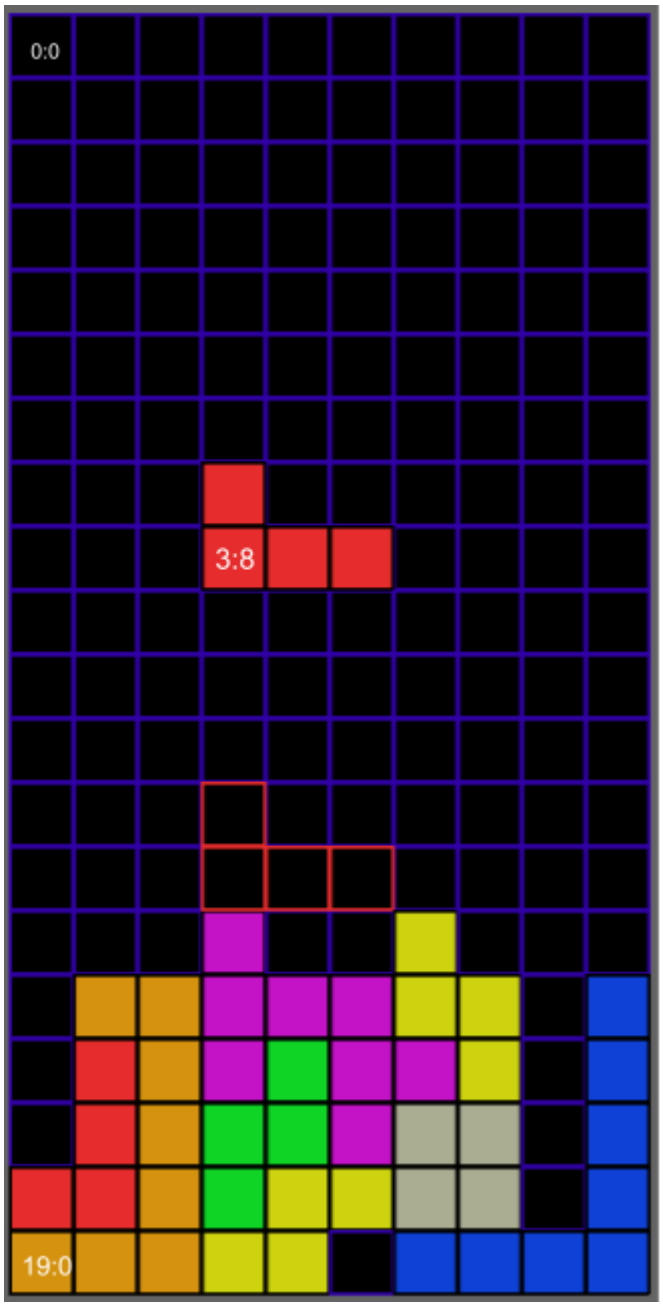
Той проверява вече изместения тетромино дали се блъска с нещо.
С пода е елементарна проверка world.GAME_HEIGHT е височината на игралното поле.
С стените също елементарно world.GAME_WIDTH е широчината игралното поле. В нашия случай те са:

```
GAME_WIDTH = 10
GAME_HEIGHT = 20
```



Последния if, при него проверяваме дали тетрогиното е стигнало вече съществуващи натрупани/застинали/неподвижни/статични тетроминота, те реално не се съхраняват като Tetromino класове, а тази информация се държи в world.grid. Той е двуизмерен масив 20x10 в нашия случай, държащ информация за цвят с триизмерен кортеж за всеки координат от полето.
Пример:
Grid[0][0] = Black
Grid[0][19] = Orange

Grid[3][8] = Black(не е Red защото там няма статично/натрупано/неподвижни тетромино)



Чрез това представяне в grid, можем да стигнем до извода че където цвета не е черно, има застинало тетронимо с което може да се блъснем.
Нека се върнем на if-а

```
# Reach locked tetrominos solid grid
if (tile[0] >= 0 and tile[1] >= 0):
    if world.grid[tile[1]][tile[0]] != world.GRID_COLOR_FILL:
        return True
```

Именно тази проверка се прави тук. Ако с координатите на квадратчето като индекс в grid не е Black, значи има колизия.

```
GRID_COLOR_FILL = (0, 0, 0)
```

В нашия случай използваме реално Black, но може да бъде всеки друг цвят за цвета на празното поле. НО!!! Не бива да бъден равен на цвета на което и да е тетромино тъй като тази логика тук няма да работи. Това не е действително потенциален бъг тъй като, чисто стилистично, тетроминотата трябва да са с друг цвят ако искаме да ги виждаме върху фона.

```
    def move(self, x, y):
        prev_tiles = self.tiles.copy()
        self.move_no_coll(x, y)

        if self.check_collision():
            self.tiles = prev_tiles
            # raised reached floor event
            if (y >= 1):
                pygame.event.post(pygame.event.Event(REACHED_FLOOR))

        sound.play_sound("movement")
```

Ако има колизия се присвоява старата позиция преди движението. Допълнително. ако движение е надолу(y >= 1), може да се допусне че сблъсъка не е с стените а с пода или статичните тетромината долу, и се поства custom event който се консумира в main loop-а, това се разглежда в следващата част

3. Събиране на тетроминота на дъното

```
        elif event.type == tetromino.REACHED_FLOOR:
            t = tetromino.lock_tetromino(t)
            w.check_for_filled_row()

def lock_tetromino(tetromino):
    for tile in tetromino.tiles:
        if tile[1] < 0:
            print("GAME OVER")
            sound.play_sound("game_over")
            pygame.time.delay(2000)
            exit(0)
        world.grid[tile[1]][tile[0]] = tetromino.color;
```

```
    return get_tetromino()
```

3.1. Условие за край на игра

Тетрминото е стигнало дъното, трябва да се направи застинало и да се появи ново тетронимо. В for цикъла първият if е условието за край на играта, ако трябва да се застине тетронимото, но някой от квадратчетата са над полето(y < 0), край на играта.

3.2. Чистене на редове

Редът:

```
world.grid[tile[1]][tile[0]] = tetromino.color;
```

Извършва застиването по вече споменатата логика на world.grid.

```
return get_tetromino()
```

Вече разгледан метод, взема следващото тетромино което трябва да се появи и го връща в main loop-a.

```
while running:
...
        elif event.type == tetromino.REACHED_FLOOR:
            t = tetromino.lock_tetromino(t)
            w.check_for_filled_row()
```

И вече то е активното тетромино.

w.check_for_filled_row() ще извърши чистенето на ред ако е пълен с застинали тетроминота

Накратко за клас World, единствено съдържа screen променлива необходима за рисуването, което не разглеждаме сега. Реално може да се премахме и да се замени с променлива world.screen която да се сетва от main-а тъй като и без това функционалността на полето е реализирана чрез променливата world.grid извън класа.

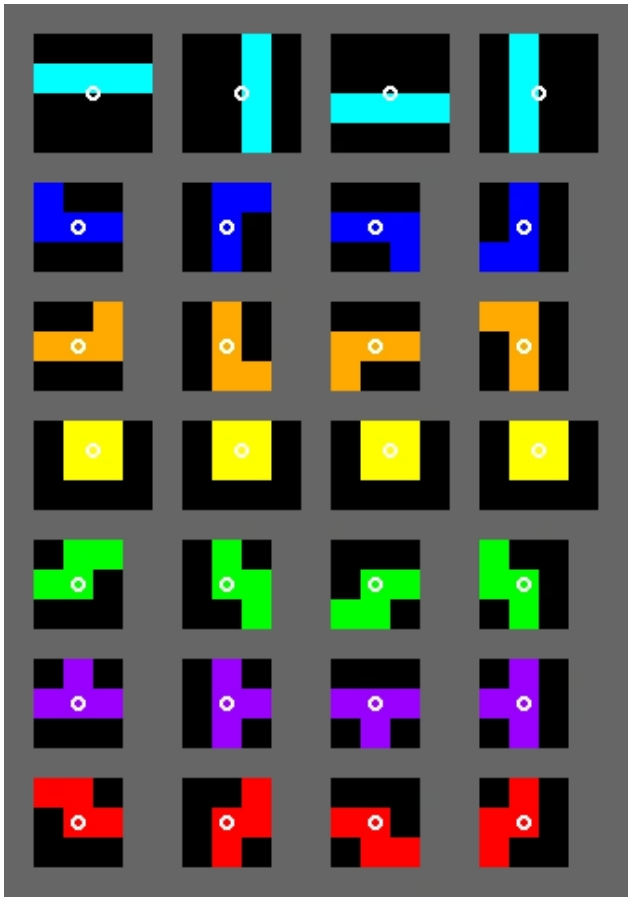
```
class World:
    screen = None
    def __init__(self, screen):
        self.screen = screen

    def check_for_filled_row(self):
        for y in range(0, GAME_HEIGHT):
            filled_row = True
            for x in range(0, GAME_WIDTH):
                if grid[y][x] == GRID_COLOR_FILL:
```

```
                filled_row = False
        if filled_row:
            for y1 in range(y - 1, -1, -1):
                for x1 in range(0, GAME_WIDTH):
                    grid[y1 + 1][x1] = grid[y1][x1]
            sound.play_sound("rotation")
```

Итерерира се през всеки ред, ако не се намери черно квадратче тоест grid[y][x] = GRID_COLOR_FILL, този ред е пълен и следва да бъде премахнат.
Премахването е “падане” на горните редове, от реда над пълния с y1 до 0 ред, всеки долен ред се замества с горния.

2.3 Въртене

Въртенето го разглеждаме чак сега въпреки че спада към точка **2. Движение,** тъй като е по-сложно и знание за статичните/застиналите тетроминота бе необходимо.

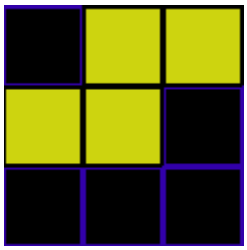


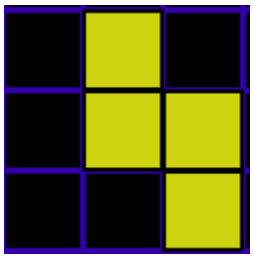
Много полезна картинка показваща желаното въртене.
Нека да се върнем за момент на листа spawns

```
spawns = [
```

```
    [(1, 1), (2, 1), (0, 1), (3, 1)],    # I
    [(1, 1), (0, 1), (1, 0), (2, 1)],    # T
    [(1, 1), (0, 1), (0, 0), (2, 1)],    # J
    [(1, 1), (0, 1), (2, 1), (2, 0)],    # L
    [(1, 1), (0, 1), (1, 0), (2, 0)],    # Z
    [(1, 1), (1, 0), (0, 0), (2, 1)],    # S
    [(1, 1), (2, 0), (1, 0), (2, 1)],    # O
]
```

Z - тетромино, вижда се че първото квадратче в списъка винаги е (1, 1),
идеята е че винаги разчитаме че първото да е централното.


Ако ще се върти по часовника тоест положително въртене(+1), се очаква да стане така.

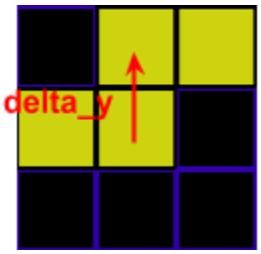


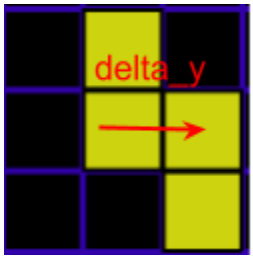
Елементарно въртене:

```
def rotate(self, rotation):
...
center_tile_index = 0
for i in range(len(self.tiles)):
            if i == center_tile_index:
                continue

            center_tile = self.tiles[center_tile_index]
            outer_tile = self.tiles[i]
            delta_x = (center_tile[0] - outer_tile[0]) * rotation
```

```
delta_y = (center_tile[1] - outer_tile[1]) * rotation

self.tiles[i] = (center_tile[0] + delta_y, center_tile[1] - delta_x)
```

Използва се разстоянието на всяко нецентрално квадратче от централното delta_x, delta_y. И след въртене delta_x реално трябва да стане все едно delta_y за даденото квадратче. Пример: квадратчето над централното (1,0) е с delta_y = 1, след въртене, в зависимост посока, новата x стойност трябва да стане x стойността на централното +/- delta_y. И анологично за новата y стойност.

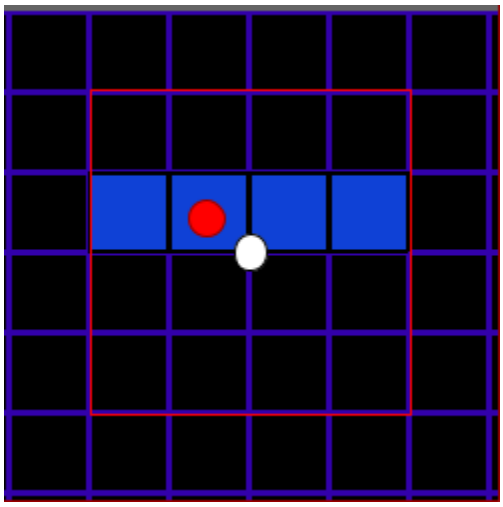




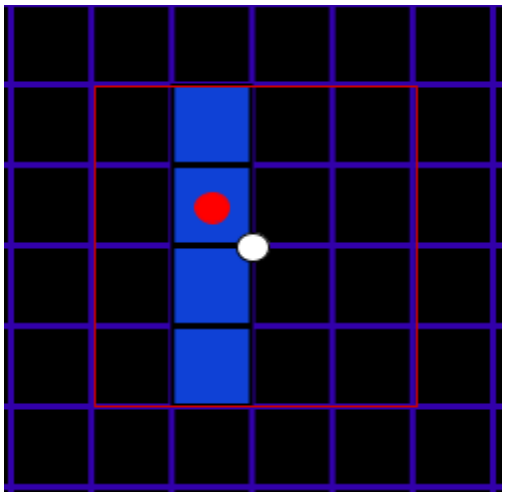
За I тетроминото има допълнителна логика тъй като няма централно квадратче. Въртенето трябва да е около ръб.

Ефектът може да се получи използвайки нормалното въртене около квадрат, но след това с допълнително изместване вертикално или хоризонтално.

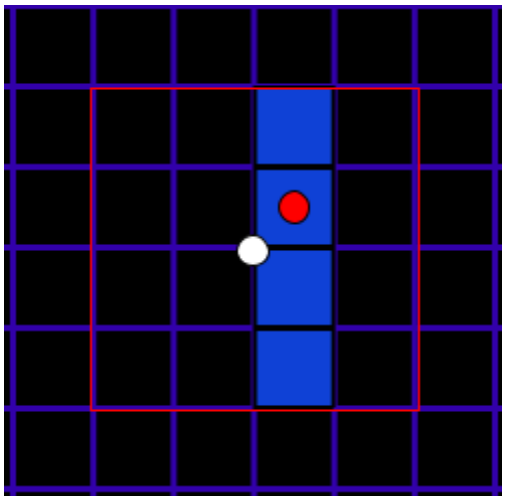
Преди въртене около централно квадратче с червената точка



След въртене около червената точка по часовника



След изместване, надясно в този случай.



Вижда се че крайният резултат е сякаш се е завъртяло около бялата точка, по часовника, което бе целта.

```
def rotate(self, rotation):
        # Block shaped 2x2 tetromino does not rotate
        if self.type == 6:
            return
        # https:#tetris.fandom.com/wiki/SRS?file=SRS-pieces.png
        # Special shift for when rotation "I" tetromino
        if self.type == 0:
            shift_x = 0
            shift_y = 0

            # If vertical before rotation
            if self.tiles[0][0] == self.tiles[1][0]:
                # If centertile above geometric center
                if self.tiles[0][1] < self.tiles[1][1]:
                    shift_y = rotation
                else:
                    shift_y = -rotation
            # If horizontal before rotation
            else:
                # If center to the left of geometric center
                if self.tiles[0][0] < self.tiles[1][0]:
                    shift_x = rotation
                else:
                    shift_x = -rotation

            self.move_no_coll(shift_x, shift_y)
```

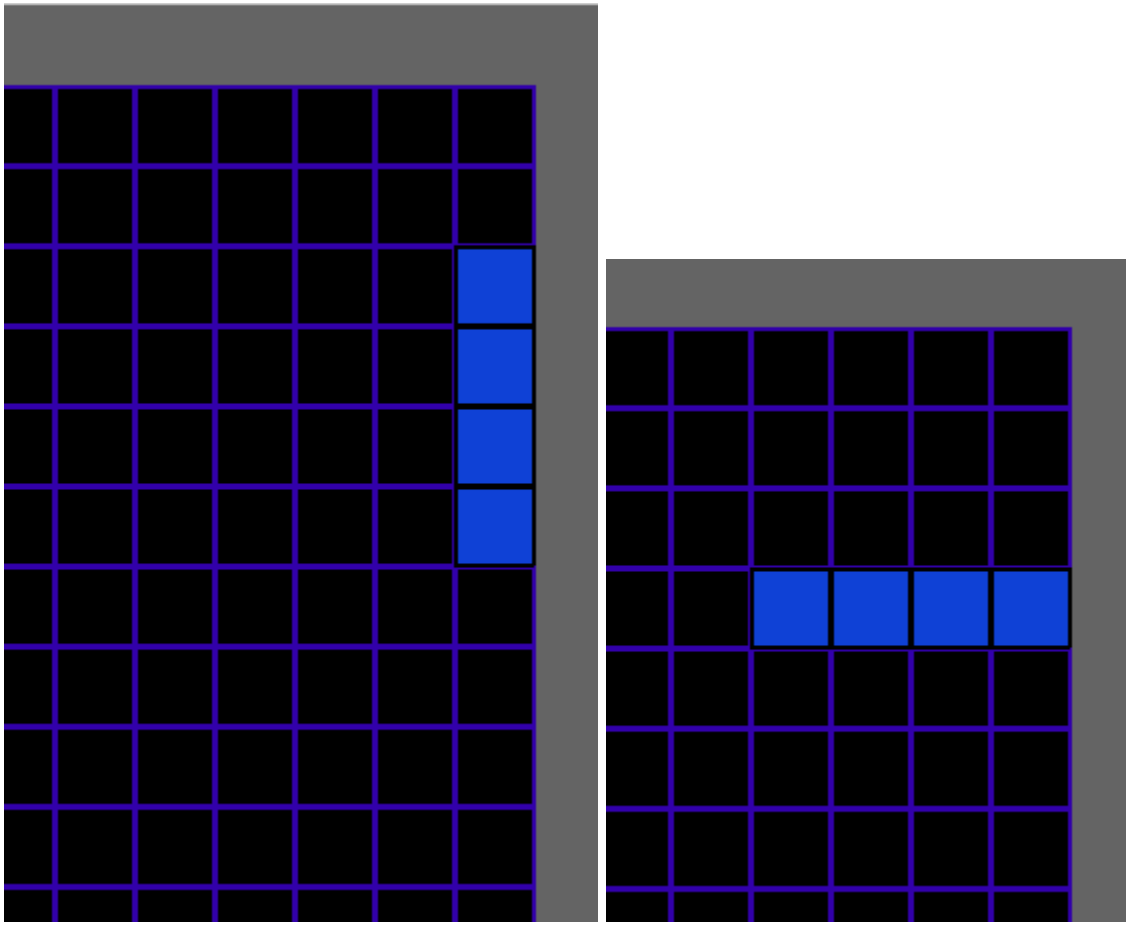
Това е кода за изместването на I тетроминота, изпълнява се преди елементарното въртене. Също в началото се вижда че квадратните О-образни тетромината не се въртят изобщо, тъй като са симетрична фигура.

Това дотук е същността на въртенето. Но възниква един въпрос, какво правим, ако при въртене тетроминото се сблъска с статична фигура или стените. Лесно решение е да не се позволява въртене в този случай. Но в https://tetris.fandom.com/wiki/SRS е написано че в такива случай се извършва изместване, това се нарича wall kick.

Wall Kicks

When the player attempts to rotate a tetromino, but the position it would normally occupy after basic rotation is obstructed, (either by the wall or floor of the playfield, or by the stack), the game will attempt to "kick" the tetromino into an alternative position nearby. Some points to note:

Следва да влезе в стената ако се завърти.



След завъртане се измества до позиция без сблъскване.

Имлементирах вече дадената логика в уикипедиата:

- When a rotation is attempted, 5 positions are sequentially tested (inclusive of basic rotation); if none are available, the rotation fails completely.
- Which positions are tested is determined by the initial rotation state, and the desired final rotation state. Because it is possible to rotate both clockwise and counter-clockwise, for each of the 4 initial states there are 2 final states. Therefore

there are a total of 8 possible rotations for each tetromino and 8 sets of wall kick data need to be described.

- The positions are commonly described as a sequence of (x, y) kick values representing translations relative to basic rotation; a convention of positive x rightwards, positive y upwards is used, e.g. (-1, 2) would indicate a kick of 1 cell left and 2 cells up.
- The *J*, *L*, *S*, *T* and *Z* tetrominoes all share the same kick values, the *I* tetromino has its own set of kick values, and the *O* tetromino does not kick.
- Several different conventions are commonly used for the naming of the rotation states. On this page, the following convention will be used:
    - 0 = spawn state
    - 1 = state resulting from a clockwise rotation ("right") from spawn
    - 2 = state resulting from 2 successive rotations in either direction from spawn.
    - 3 = state resulting from a counter-clockwise ("left") rotation from spawn

Накратко има състояния на въртене, 0 е систоянието при появяване, ако се въртим по часовника +1 ако обратно -1, точно като часовник от 0 до 3.

За всяко възможно въртене от едно състояние до друго има възможни измествания, пробва се всяко докато не се намери подходящо(без колизии).

J, L, S, T, Z Tetromino Wall Kick Data

	Test 1	Test 2	Test 3	Test 4	Test 5
0>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
1>>0	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
1>>2	(0, 0)	(1, 0)	(1,-1)	(0, 2)	(1, 2)
2>>1	(0, 0)	(-1, 0)	(-1, 1)	(0,-2)	(-1,-2)
2>>3	(0, 0)	(1, 0)	(1, 1)	(0,-2)	(1,-2)
3>>2	(0, 0)	(-1, 0)	(-1,-1)	(0, 2)	(-1, 2)
3>>0	(0, 0)	(-1, 0)	(-1,-1)	(0, 2)	(-1, 2)
0>>3	(0, 0)	(1, 0)	(1, 1)	(0,-2)	(1,-2)

I Tetromino Wall Kick Data

	Test 1	Test 2	Test 3	Test 4	Test 5
0>>1	(0, 0)	(-2, 0)	(1, 0)	(-2,-1)	(1, 2)
1>>0	(0, 0)	(2, 0)	(-1, 0)	(2, 1)	(-1,-2)
1>>2	(0, 0)	(-1, 0)	(2, 0)	(-1, 2)	(2,-1)
2>>1	(0, 0)	(1, 0)	(-2, 0)	(1,-2)	(-2, 1)
2>>3	(0, 0)	(2, 0)	(-1, 0)	(2, 1)	(-1,-2)
3>>2	(0, 0)	(-2, 0)	(1, 0)	(-2,-1)	(1, 2)
3>>0	(0, 0)	(1, 0)	(-2, 0)	(1,-2)	(-2, 1)
0>>3	(0, 0)	(-1, 0)	(2, 0)	(-1, 2)	(2,-1)

```
kick_data = [
    [(0, 0), (1, 0), (1, -1), (0, 2), (1, 2)],       # 0 -> 3
    [(0, 0), (-1, 0), (-1, -1), (0, 2), (-1, 2)],    # 0 -> 1
    [(0, 0), (1, 0), (1, 1), (0, -2), (1, -1)],      # 1 -> 0
    [(0, 0), (1, 0), (1, 1), (0, -2), (1, -2)],      # 1 -> 2
    [(0, 0), (-1, 0), (-1, -1), (0, 2), (-1, 2)],    # 2 -> 1
    [(0, 0), (1, 0), (1, -1), (0, 2), (1, 2)],       # 2 -> 3
    [(0, 0), (-1, 0), (-1, 1), (0, -2), (-1, -2)],   # 3 -> 2
    [(0, 0), (-1, 0), (-1, -1), (0, -2), (-1, -2)]   # 3 -> 0
```

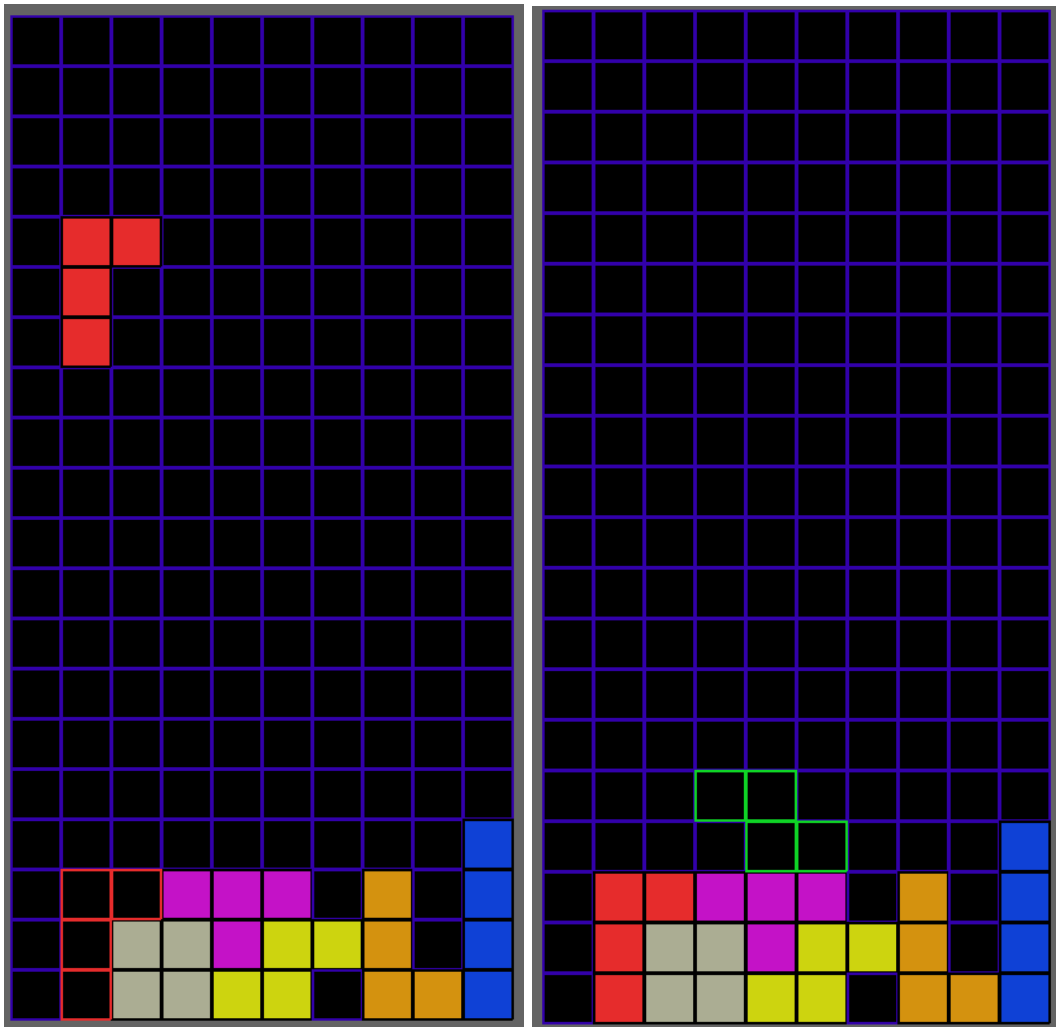
```
]

# Special kickdata from "I" tetromino
kick_data_I =[
    [(0, 0), (-1, 0), (2, 0), (-1, -2), (2, 1)],     # 0 -> 4
    [(0, 0), (-2, 0), (1, 0), (-2, 1), (1, -2)],     # 0 -> 1
    [(0, 0), (2, 0), (-1, 0), (2, 1), (-1, -2)],     # 1 -> 0
    [(0, 0), (-1, 0), (2, 0), (-1, -2), (2, -1)],    # 1 -> 2
    [(0, 0), (1, 0), (2, 0), (1, 2), (-2, 1)],       # 2 -> 1
    [(0, 0), (2, 0), (-1, 0), (2, -1), (-1, 2)],     # 2 -> 3
    [(0, 0), (-2, 0), (1, 0), (-2, 1), (1, -2)],     # 3 -> 2
    [(0, 0), (1, 0), (2, 0), (1, -2), (-2, 1)]       # 3 -> 0
]
```

Ето кодът в програмата същото като https://tetris.fandom.com/wiki/SRS, НО у стойностите са противоположни тъй като в сайта се използва положителен у нагоре.

```
# Wallkicks https://tetris.fandom.com/wiki/SRS go to "Wall Kicks" secton
        rotate_from = self.rotation_state
        rotate_to = self.rotation_state + rotation
        # Like a clock logic
        if rotate_to > 3:
            rotate_to = 0
        elif rotate_to < 0:
            rotate_to = 3

        # The offsets to be tried to avoid collision
        offsets = get_wallkick_offsets(rotate_from, rotate_to, self.type)
        orig_tiles = self.tiles.copy()
        for offset in offsets:
            print("From: ", rotate_from, "To: ", rotate_to)
            print("Trying offset: ", offset)
            self.move_no_coll(offset[0], offset[1])
            if self.check_collision():
                self.tiles = orig_tiles.copy()
            else:
                # Succefully rotated with no collision
                self.rotation_state = rotate_to
                sound.play_sound("rotation")
                return

        print("ERROR: ROTATION FAILED")
```

2.4. Хард-дроп

Когато играчът, вместо да чака тетроминото автоматично да стигне дъното, реши да натисне бутон който прави това незабавно.



main.py

```
while running:
    for event in pygame.event.get():
        ...
        elif event.type == pygame.KEYDOWN:
            ...
            elif event.key == pygame.K_SPACE:
                t = tetromino.hard_drop_tetronimo(t)
                w.check_for_filled_row()
                sound.play_sound("hard_drop")
```

tetromino.py

```
def hard_drop_tetronimo(t):
    t = get_ghost_tetromino(t)
    return lock_tetromino(t)
```

```
def get_ghost_tetromino(t):
    t = copy.deepcopy(t)
    t.is_ghost = True
    while not t.check_collision():
        t.move_no_coll(0, 1)

    t.move_no_coll(0, -1)

    return t
```

Ghost тетромино се генерира като се вземе копие на сегашното контролирано тетромино, мести се това копие докато надолу докато не се стигне колизия.
За да стане хард-дропа просто се вика lock_tetromino на това ghost тетромино.

Рисуване

То става в main цикъла за всеки кадър.

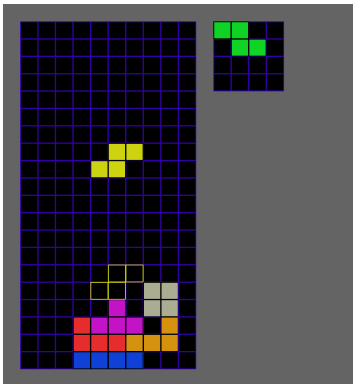
```
while running:
    ...
    screen.fill(world.SCREEN_COLOR)
    w.render()
    t.render()
    ghost = tetromino.get_ghost_tetromino(t)
    ghost.render()
    gui.show_next_tetromino(screen)

    pygame.display.flip()
    clock.tick(60)
```

Последните два реда: обновява се екрана и се блокира изпълнението колкото е необходимо, за постигане на дадените кадри в секунда, 60 в този случай

Ще разгледаме един по един всеки ред какво рисува:

```
screen.fill(world.SCREEN_COLOR)
```

Стандартен метод на pygame. Просто се запълва целия screen с сиво в нашия случай и отгоре се рисува всичко друго, затова се извиква първо.



```
class World:
      ...
      def render(self):
       for y in range(0, GAME_HEIGHT):
           for x in range(0, GAME_WIDTH):
               r = pygame.Rect((game_start_x + TILE_SIZE*x, game_start_y +
TILE_SIZE*y), (TILE_SIZE, TILE_SIZE))
               color = grid[y][x]
               pygame.draw.rect(self.screen, color, r)
               if color != GRID_COLOR_FILL:
                   pygame.draw.rect(self.screen, tetromino.BORDER_COLOR, r, 2)
               else:
                   pygame.draw.rect(self.screen, GRID_COLOR_BORDER, r, 2)
```

Рисува се 20x10 полето

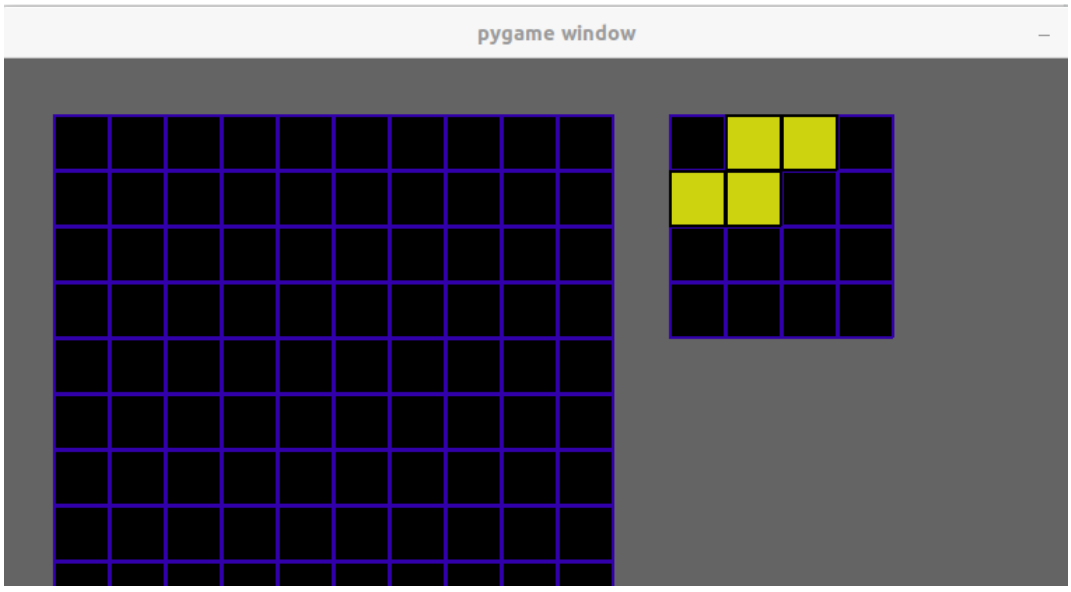
```
class Tetromino:
      ...
      def render(self):
       for tile in self.tiles:
           if tile[1] < 0:
               continue;
           r = pygame.Rect((world.game_start_x + world.TILE_SIZE*tile[0],
world.game_start_y + world.TILE_SIZE*tile[1]),\
                (world.TILE_SIZE, world.TILE_SIZE))
           if (not self.is_ghost):
               pygame.draw.rect(screen, self.color, r)
               pygame.draw.rect(screen, BORDER_COLOR, r, 2)
           else:
               pygame.draw.rect(screen, self.color, r, 2)
```

Рисува се активното тетромино както и ghost-тетроминото чрез този метод.

```
gui.show_next_tetromino(screen)
```

gui.py

```
import pygame

import tetromino
import world

GUI_PLACEMENT = (world.GAME_WIDTH + 2, world.GAME_PLACEMENT[1])
gui_start_x = GUI_PLACEMENT[0] * world.TILE_SIZE
gui_start_y = GUI_PLACEMENT[1] * world.TILE_SIZE

def show_next_tetromino(screen):
   if len(tetromino.tetromino_set) > 0:
       type = tetromino.tetromino_set[0].type

       # Render a grid
       for y in range(0, 4):
           for x in range(0, 4):
               r = pygame.Rect((gui_start_x + world.TILE_SIZE*x, gui_start_y +
world.TILE_SIZE*y)\
               ,(world.TILE_SIZE, world.TILE_SIZE))

               pygame.draw.rect(screen, world.GRID_COLOR_FILL, r)
               pygame.draw.rect(screen, world.GRID_COLOR_BORDER, r, 2)

       # Render tetromino
       tiles = tetromino.spawns[type]
       for tile in tiles:
           r = pygame.Rect((gui_start_x + world.TILE_SIZE*tile[0], gui_start_y +
world.TILE_SIZE*tile[1])\
               ,(world.TILE_SIZE, world.TILE_SIZE))

           pygame.draw.rect(screen, tetromino.colors[type], r)
           pygame.draw.rect(screen, tetromino.BORDER_COLOR, r, 2)
```



Рисува следващото тетромино което ще се появи в малкото поле горе дясно. Използва tetromino.tetromino_set[0].type, тоест първи елемент в списъка за появяване в tetromino модулът, за да разбере типът на следващото тетромино. И tetromino.spawns за да разбере как да го нарисува.

Звук

sound.py

```
import pygame

pygame.mixer.music.set_volume(0.03)

sounds = {
    "movement" : pygame.mixer.Sound("sfx/movement.wav"),
    "rotation" : pygame.mixer.Sound("sfx/rotation.wav"),
    "line_clear" : pygame.mixer.Sound("sfx/line_clear.wav"),
    "hard_drop" : pygame.mixer.Sound("sfx/hard_drop.wav"),
    "game_over" : pygame.mixer.Sound("sfx/game_over.wav"),
    "pause" : pygame.mixer.Sound("sfx/pause.wav"),
}

def play_music():
    pygame.mixer.music.load("sfx/tetris.wav")
    pygame.mixer.music.play(-1)

def stop_music():
    pygame.mixer.music.stop()
```

```
def play_sound(sound):
    play_sound = sounds[sound]
    play_sound.set_volume(0.5)
    pygame.mixer.Sound.play(play_sound)
```

play_music() се вика в началото преди влизане в main цикъла, изпълява безкрайно музиката чрез повторение.
play_stound() с даден аргумент се вика в адекватните моменти и изпълнява еднократен къс звук.