Final Project Report

Tetris

I. Game Overview

Genre: Puzzle Game

Concept: The player controls the blocks that are falling from the top of the screen and will try to fill in the blank spots on the bottom of the board which will then remove said blocks, if the blocks stack up until the top of the board, the player loses.

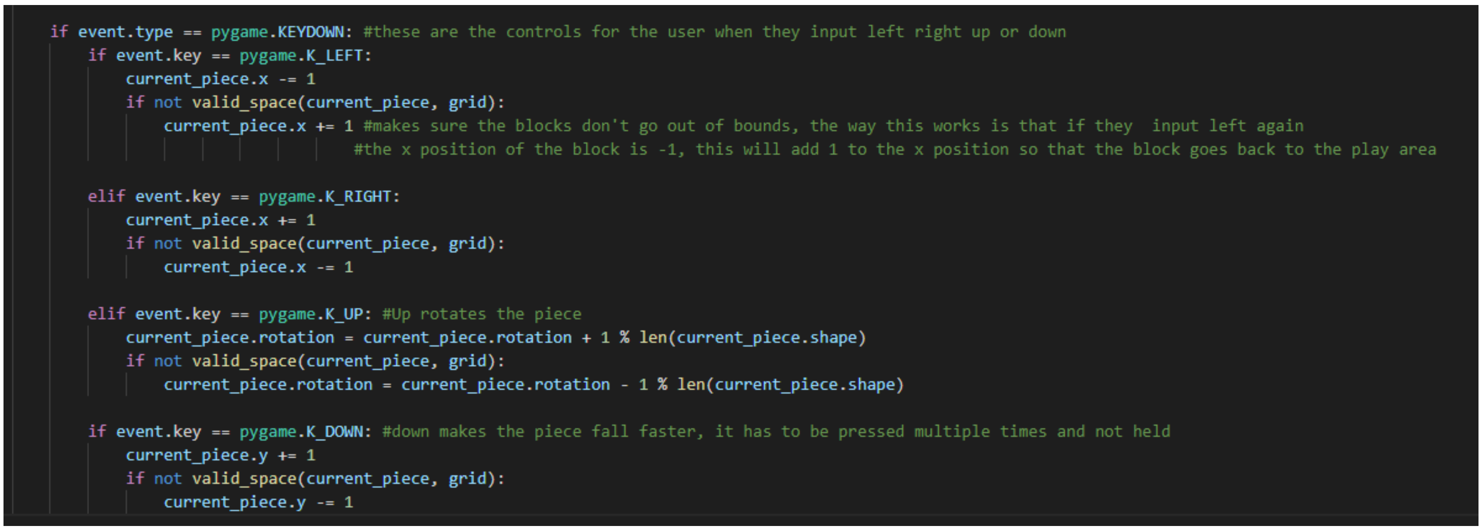
Visual: 8-bit

Game Flow: The game starts off slow, but slowly gets faster and faster as time goes on.

II. Gameplay

Objective: The objective of the game is to place blocks in a horizontal line and keep going for as long as possible

Controls: The player can rotate the blocks by pressing the Up arrow key, move the block left by pressing the left arrow key, move the block right by pressing the right arrow key and making the block fall down faster by tapping the down key. This is done through this function

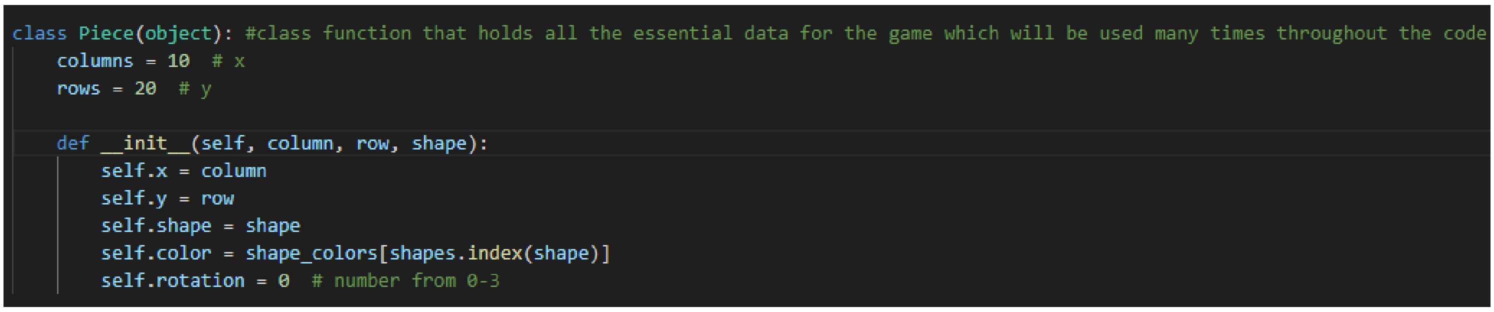


```python
if event.type == pygame.KEYDOWN: #these are the controls for the user when they input left right up or down
    if event.key == pygame.K_LEFT:
        current_piece.x -= 1
        if not valid_space(current_piece, grid):
            current_piece.x += 1 #makes sure the blocks don't go out of bounds, the way this works is that if they  input left again
                                 #the x position of the block is -1, this will add 1 to the x position so that the block goes back to the play area

    elif event.key == pygame.K_RIGHT:
        current_piece.x += 1
        if not valid_space(current_piece, grid):
            current_piece.x -= 1

    elif event.key == pygame.K_UP: #Up rotates the piece
        current_piece.rotation = current_piece.rotation + 1 % len(current_piece.shape)
        if not valid_space(current_piece, grid):
            current_piece.rotation = current_piece.rotation - 1 % len(current_piece.shape)

    if event.key == pygame.K_DOWN: #down makes the piece fall faster, it has to be pressed multiple times and not held
        current_piece.y += 1
        if not valid_space(current_piece, grid):
            current_piece.y -= 1
```

III. Interface

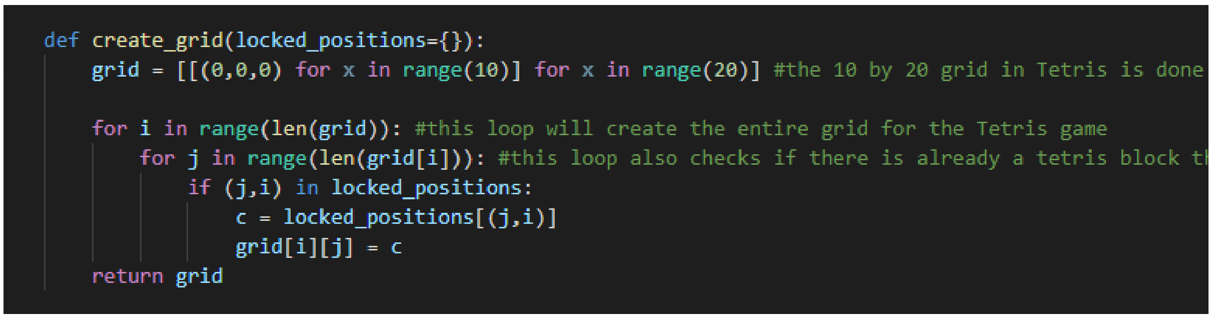
As Tetris is a fairly simple game without very many mechanics and only operates on one general interface, the project doesn't necessarily have that many Interfaces. However, these are the codes that make up the interface

- Class function that will be used repeatedly throughout the code



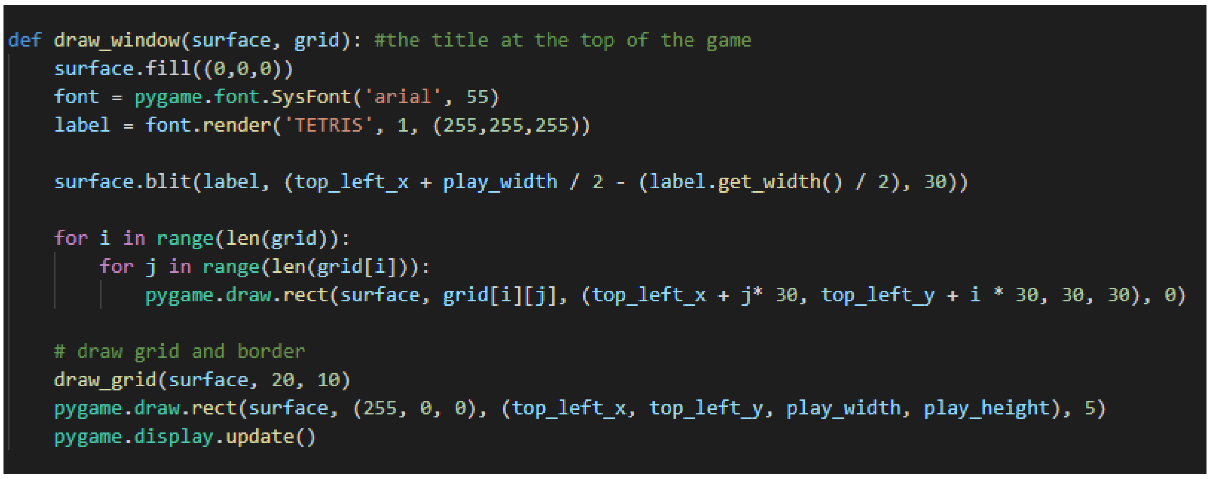
```python
class Piece(object): #class function that holds all the essential data for the game which will be used many times throughout the code
    columns = 10  # x
    rows = 20  # y

    def __init__(self, column, row, shape):
        self.x = column
        self.y = row
        self.shape = shape
        self.color = shape_colors[shapes.index(shape)]
        self.rotation = 0  # number from 0-3
```

- Grid function that is used for drawing the in-game grid, it also includes the functions that locks the blocks that fall to the bottom in place



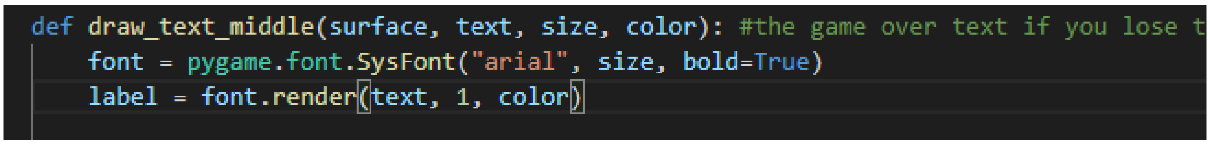
```python
def create_grid(locked_positions={}):
    grid = [[(0,0,0) for x in range(10)] for x in range(20)] #the 10 by 20 grid in Tetris is done

    for i in range(len(grid)): #this loop will create the entire grid for the Tetris game
        for j in range(len(grid[i])): #this loop also checks if there is already a tetris block t
            if (j,i) in locked_positions:
                c = locked_positions[(j,i)]
                grid[i][j] = c
    return grid
```

- Title screen and board that the game takes place in



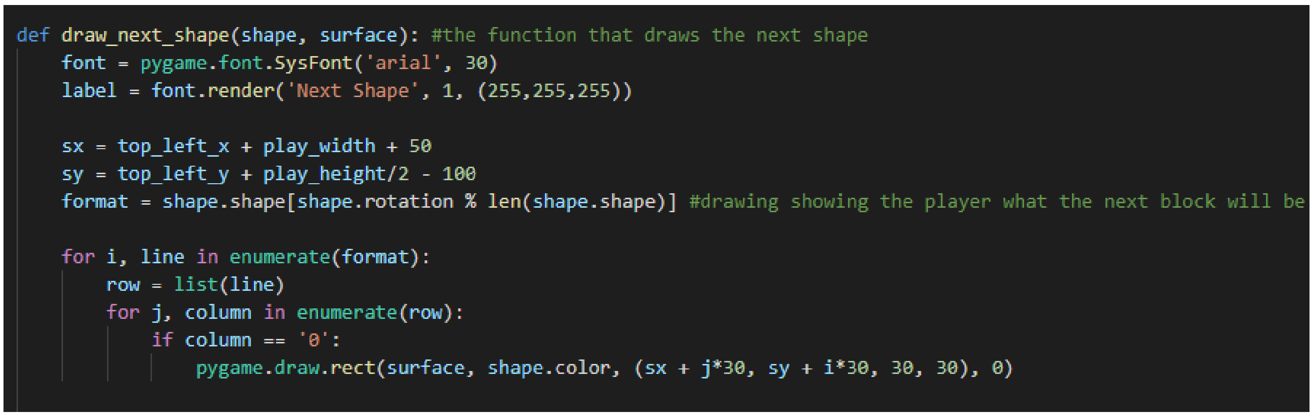
```python
def draw_window(surface, grid): #the title at the top of the game
    surface.fill((0,0,0))
    font = pygame.font.SysFont('arial', 55)
    label = font.render('TETRIS', 1, (255,255,255))

    surface.blit(label, (top_left_x + play_width / 2 - (label.get_width() / 2), 30))

    for i in range(len(grid)):
        for j in range(len(grid[i])):
            pygame.draw.rect(surface, grid[i][j], (top_left_x + j* 30, top_left_y + i * 30, 30, 30), 0)

    # draw grid and border
    draw_grid(surface, 20, 10)
    pygame.draw.rect(surface, (255, 0, 0), (top_left_x, top_left_y, play_width, play_height), 5)
    pygame.display.update()
```

- Game over text format that will be used later on



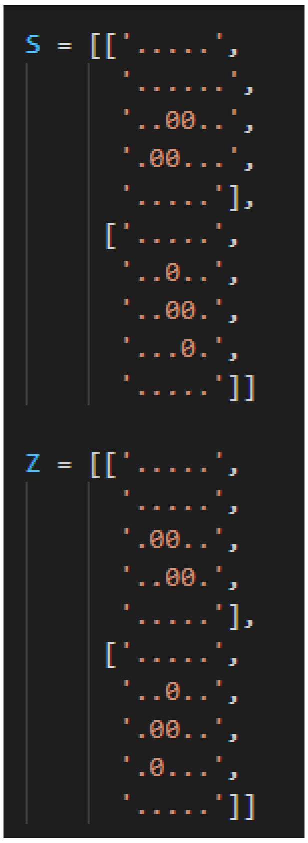
```python
def draw_text_middle(surface, text, size, color): #the game over text if you lose t
    font = pygame.font.SysFont("arial", size, bold=True)
    label = font.render(text, 1, color)
```

- Text and drawing of what the next block shape will be

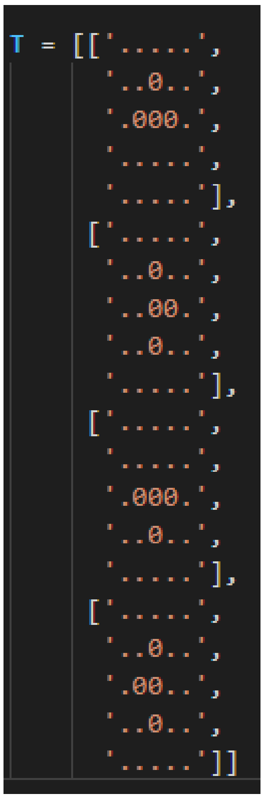


```python
def draw_next_shape(shape, surface): #the function that draws the next shape
    font = pygame.font.SysFont('arial', 30)
    label = font.render('Next Shape', 1, (255,255,255))

    sx = top_left_x + play_width + 50
    sy = top_left_y + play_height/2 - 100
    format = shape.shape[shape.rotation % len(shape.shape)] #drawing showing the player what the next block will be

    for i, line in enumerate(format):
        row = list(line)
        for j, column in enumerate(row):
            if column == '0':
                pygame.draw.rect(surface, shape.color, (sx + j*30, sy + i*30, 30, 30), 0)
```

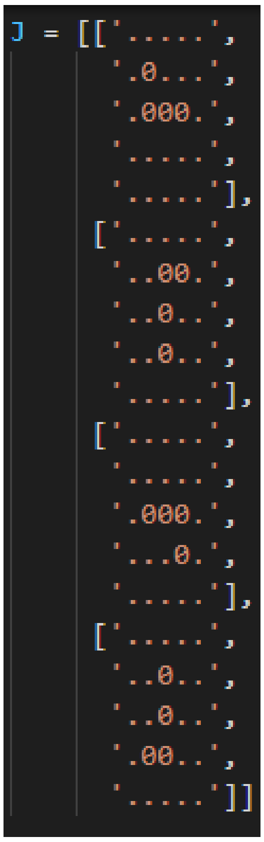
- Block Types, they are stored within a long list which will then be called upon by another function



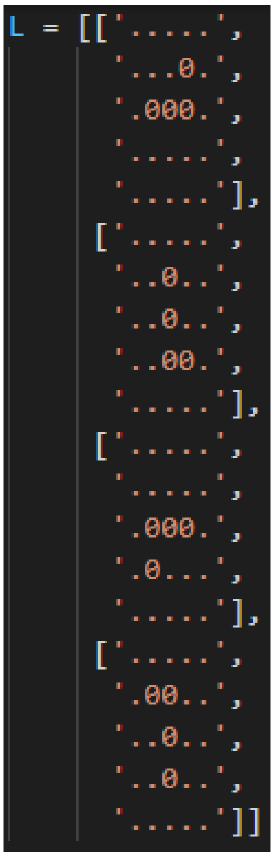
```python
S = [['.....',
      '......',
      '..00..',
      '.00...',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']]

Z = [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']]
```



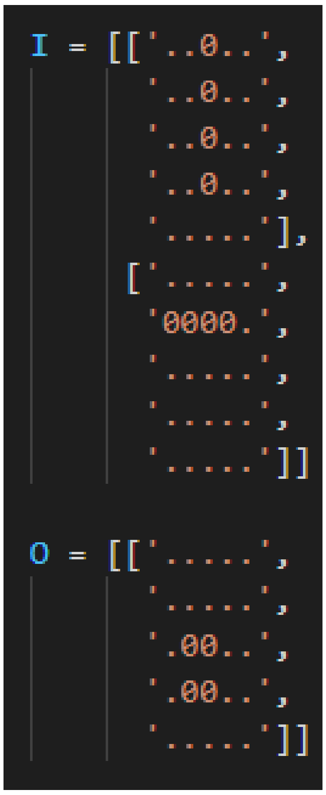
```python
T = [['.....',
      '..0..',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '..0..',
      '.....'],
     ['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '..0..',
      '.....']]
```



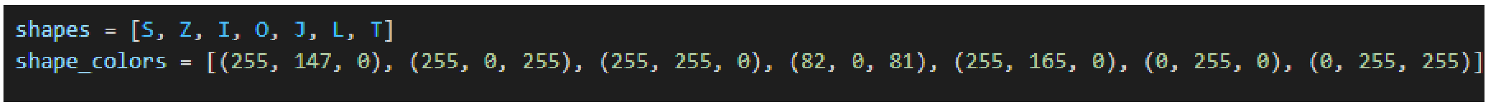
```python
J = [['.....',
      '.0...',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..00.',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '.....',
      '.000.',
      '...0.',
      '.....'],
     ['.....',
      '..0..',
      '..0..',
      '.00..',
      '.....']]
```



```python
L = [['.....',
      '...0.',
      '.000.',
      '.....',
      '.....'],
     ['.....',
      '..0..',
      '..0..',
      '..00.',
      '.....'],
     ['.....',
      '.....',
      '.000.',
      '.0...',
      '.....'],
     ['.....',
      '.00..',
      '..0..',
      '..0..',
      '.....']]
```

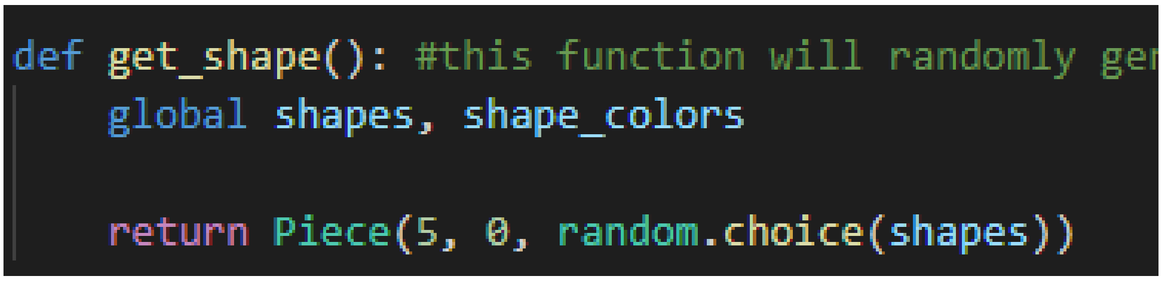


```python
I = [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']]

O = [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']]
```



```python
shapes = [S, Z, I, O, J, L, T]
shape_colors = [(255, 147, 0), (255, 0, 255), (255, 255, 0), (82, 0, 81), (255, 165, 0), (0, 255, 0), (0, 255, 255)]
```

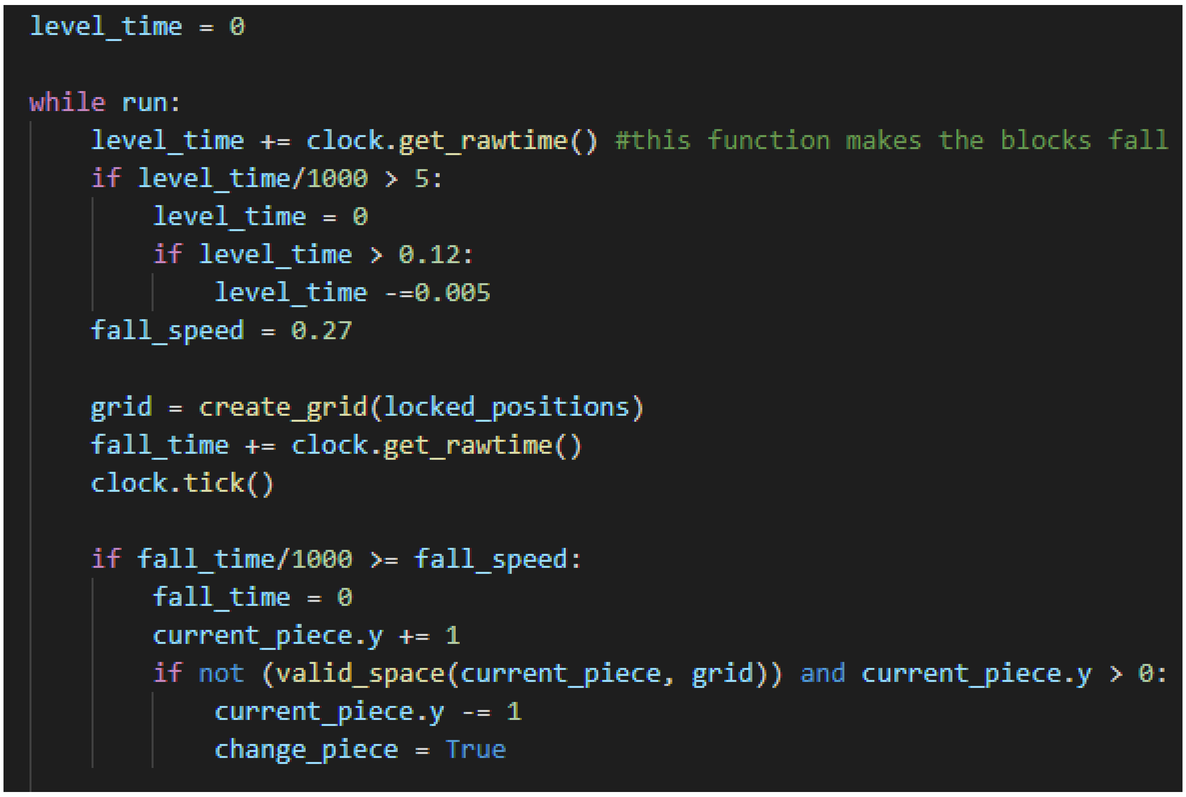
- The list will then be called by this function which will then turn them into blocks



```
def get_shape(): #this function will randomly ger
    global shapes, shape_colors

    return Piece(5, 0, random.choice(shapes))
```

IV. Gameplay

While the game is running, there is a timer that will start ticking. This will make the game run faster and faster the longer you are playing, making the blocks fall faster and thus, making the game harder. It is done through this function.



```
level_time = 0

while run:
    level_time += clock.get_rawtime() #this function makes the blocks fall
    if level_time/1000 > 5:
        level_time = 0
        if level_time > 0.12:
            level_time -=0.005
    fall_speed = 0.27

    grid = create_grid(locked_positions)
    fall_time += clock.get_rawtime()
    clock.tick()

    if fall_time/1000 >= fall_speed:
        fall_time = 0
        current_piece.y += 1
        if not (valid_space(current_piece, grid)) and current_piece.y > 0:
            current_piece.y -= 1
            change_piece = True
```