Name: Wood Block Puzzle game

Description: The Wood block puzzle game consists of blocks, of different shapes. that appear one after another. And the objective is to place these blocks on the screen such that we form rows at the bottom of the board.

Every completed row increases the score. To do so, the player will be allowed to move the falling block to left, right, down and also rotate it using the keyboard buttons.

We are done discussing the necessities of the project, let us look into the steps to be followed to build the project.

i. Create a matrix storing the information about the block and write a class to handle these details.

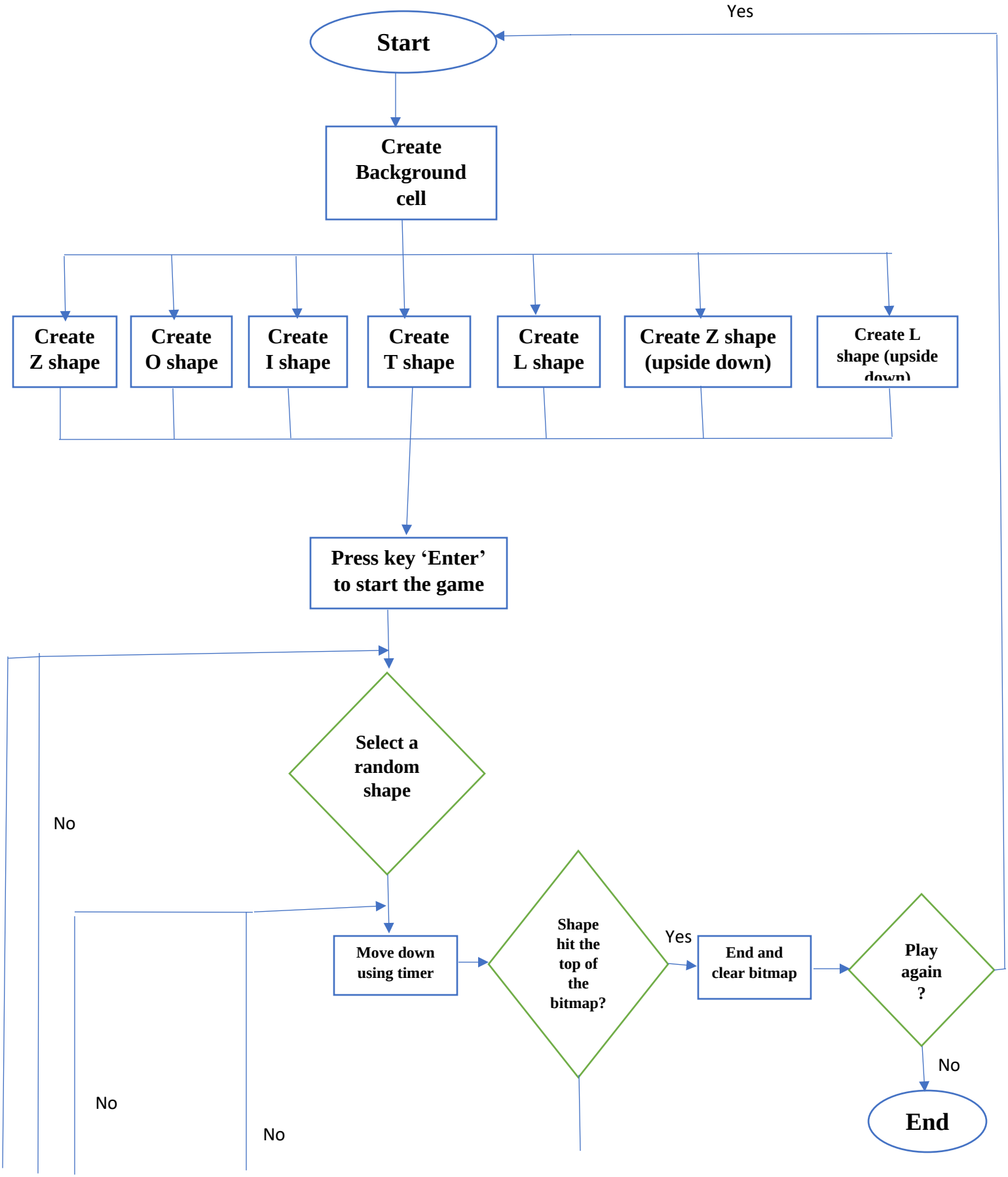
ii. Create a class to handle the creation of blocks, their movement, and placement.

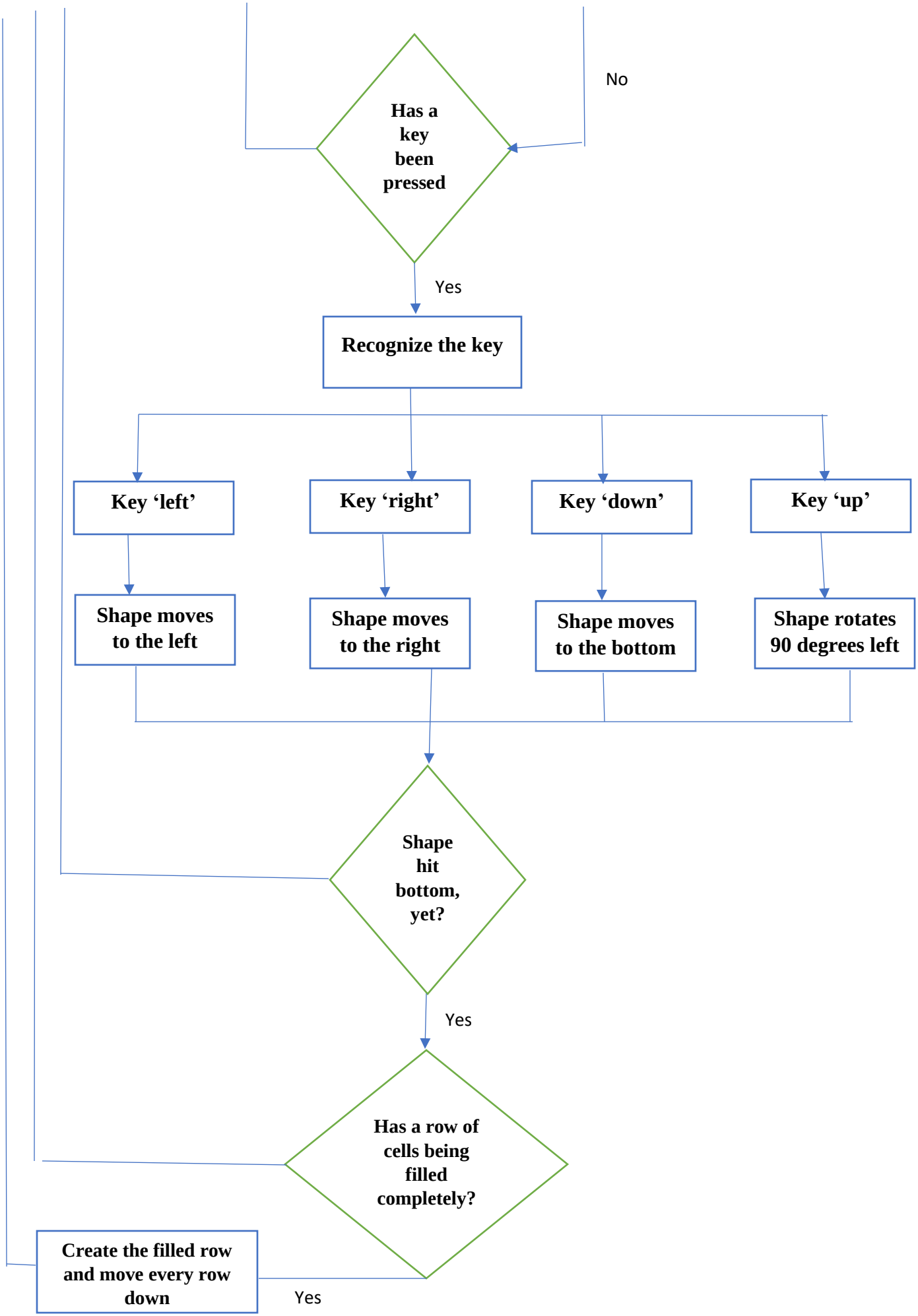
iii. Create the main window for the game.

iv. Keep checking the button pressed and the status of the blocks in the board.

v. End the game when the blocks touch the top of the board.

Flow chart: The flow chart of this game is:




Has a key been pressed
No
Yes
Recognize the key
Key ‘left’
Key ‘right’
Key ‘down’
Key ‘up’
Shape moves to the left
Shape moves to the right
Shape moves to the bottom
Shape rotates 90 degrees left
Shape hit bottom, yet?
Yes
Has a row of cells being filled completely?
Create the filled row and move every row down
Yes

Code:

```
import os
import pygame
import random
os.chdir(os.path.dirname(os.path.realpath(__file__)))

if not os.path.isfile("scores.txt"):
    with open("scores.txt", "a") as file:
        pass

pygame.font.init()

s_width = 800
s_height = 700
play_width = 300
play_height = 600
block_size = 30

top_left_x = (s_width - play_width) // 2
top_left_y = s_height - play_height

# SHAPE FORMATS
S = [['.....',
      '.....',
      '..00.',
```

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

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

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

```
        '..00.',
        '..0..',
        '..0..',
        '.....'],
       ['.....',
        '.....',
        '.000.',
        '...0.',
        '.....'],
       ['.....',
        '..0..',
        '..0..',
        '.00..',
        '.....']]

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

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

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

```
      '..0..',
      '.....']]

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

shapes = [S, Z, I, O, J, L, T]
```

```
shape_colors = [(0, 255, 0), (255, 0, 0), (0, 255, 255), (255, 255, 0), (255, 165, 0), (0, 0, 255), (128, 0, 128)]
# index 0 - 6 represent shape


class Piece(object):
    def __init__(self, x, y, shape):
        self.x = x
        self.y = y
        self.shape = shape
        self.color = shape_colors[shapes.index(shape)]
        self.rotation = 0



def create_grid(locked_pos={}):
    grid = [[(0,0,0) for _ in range(10)] for _ in range(20)]

    for i in range(len(grid)):
        for j in range(len(grid[i])):
            if (j, i) in locked_pos:
                c = locked_pos[(j,i)]
                grid[i][j] = c
    return grid



def convert_shape_format(shape):
    positions = []
```

```
    format = shape.shape[shape.rotation % len(shape.shape)]

    for i, line in enumerate(format):
        row = list(line)
        for j, column in enumerate(row):
            if column == '0':
                positions.append((shape.x + j, shape.y + i))

    for i, pos in enumerate(positions):
        positions[i] = (pos[0] - 2, pos[1] - 4)

    return positions



def valid_space(shape, grid):
    accepted_pos = [[(j, i) for j in range(10) if grid[i][j] == (0,0,0)] for i in range(20)]
    accepted_pos = [j for sub in accepted_pos for j in sub]

    formatted = convert_shape_format(shape)

    for pos in formatted:
        if pos not in accepted_pos:
            if pos[1] > -1:
                return False
    return True
```

```
def check_lost(positions):
    for pos in positions:
        x, y = pos
        if y < 1:
            return True

    return False



def get_shape():
    return Piece(5, 0, random.choice(shapes))



def draw_text_middle(surface, text, size, color):
    font = pygame.font.SysFont("comicsans", size, bold=True)
    label = font.render(text, 1, color)

    surface.blit(label, (top_left_x + play_width /2 - (label.get_width()/2), top_left_y
+ play_height/2 - label.get_height()/2))



def draw_grid(surface, grid):
    sx = top_left_x
    sy = top_left_y
```

```
    for i in range(len(grid)):
        pygame.draw.line(surface, (128,128,128), (sx, sy + i*block_size), (sx+play_width, sy+ i*block_size))
        for j in range(len(grid[i])):
            pygame.draw.line(surface, (128, 128, 128), (sx + j*block_size, sy),(sx + j*block_size, sy + play_height))



def clear_rows(grid, locked):


    inc = 0
    for i in range(len(grid)-1, -1, -1):
        row = grid[i]
        if (0,0,0) not in row:
            inc += 1
            ind = i
            for j in range(len(row)):
                try:
                    del locked[(j,i)]
                except:
                    continue


    if inc > 0:
        for key in sorted(list(locked), key=lambda x: x[1])[::-1]:
            x, y = key
```

```
            if y < ind:
                newKey = (x, y + inc)
                locked[newKey] = locked.pop(key)

    return inc


def draw_next_shape(shape, surface):
    font = pygame.font.SysFont('comicsans', 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)]

    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*block_size, sy + i*block_size, block_size, block_size), 0)

    surface.blit(label, (sx + 10, sy - 30))


def update_score(nscore):
```

```
    score = max_score()

    with open('scores.txt', 'w') as file:
        if int(score) > nscore:
            file.write(str(score))
        else:
            file.write(str(nscore))



def max_score():
    try:
        with open('scores.txt', 'r') as file:
            lines = file.readlines()
            score = lines[0].strip()
    except:
        score = "0"
    return score



def draw_window(surface, grid, score=0, last_score = 0):
    surface.fill((0, 0, 0))

    pygame.font.init()
    font = pygame.font.SysFont('comicsans', 60)
    label = font.render('Block Puzzle', 1, (255, 255, 255))
```

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

    # current score
    font = pygame.font.SysFont('comicsans', 30)
    label = font.render('Score: ' + str(score), 1, (255,255,255))

    sx = top_left_x + play_width + 50
    sy = top_left_y + play_height/2 - 100

    surface.blit(label, (sx + 20, sy + 160))
    # last score
    label = font.render('High Score: ' + last_score, 1, (255,255,255))

    sx = top_left_x - 200
    sy = top_left_y + 200

    surface.blit(label, (sx + 20, sy + 160))

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

    pygame.draw.rect(surface, (255, 0, 0), (top_left_x, top_left_y, play_width, play_height), 5)
```

```
    draw_grid(surface, grid)
    #pygame.display.update()


def main(win):
    last_score = max_score()
    locked_positions = {}
    grid = create_grid(locked_positions)

    change_piece = False
    run = True
    current_piece = get_shape()
    next_piece = get_shape()
    clock = pygame.time.Clock()
    fall_time = 0
    fall_speed = 0.27
    level_time = 0
    score = 0

    while run:
        grid = create_grid(locked_positions)
        fall_time += clock.get_rawtime()
        level_time += clock.get_rawtime()
        clock.tick()
```

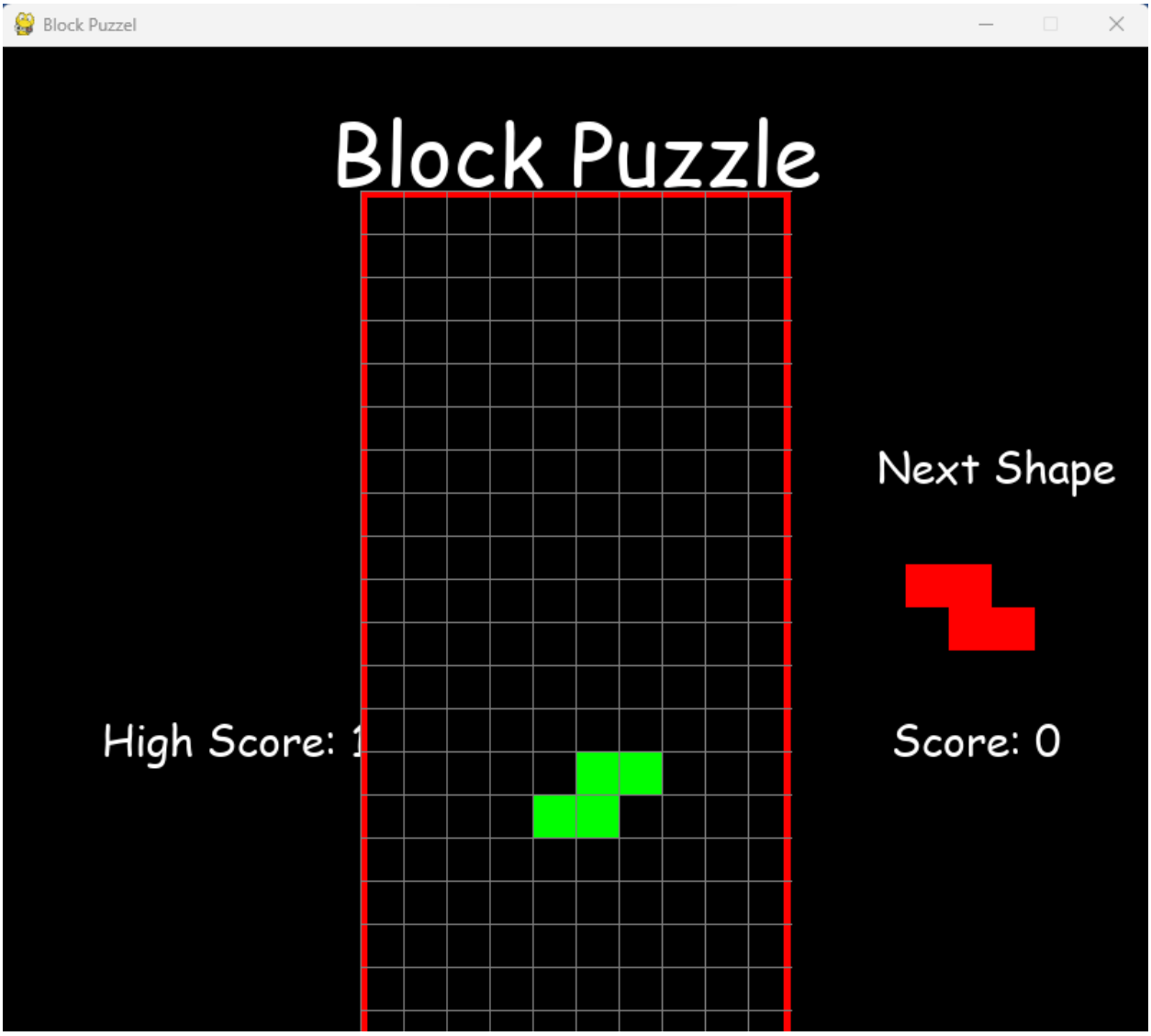
```
if level_time/1000 > 5:
    level_time = 0
    if level_time > 0.12:
        level_time -= 0.005

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

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        run = False
        pygame.display.quit()

    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_LEFT:
            current_piece.x -= 1
            if not(valid_space(current_piece, grid)):
                current_piece.x += 1
        if event.key == pygame.K_RIGHT:
            current_piece.x += 1
```

```
            if not(valid_space(current_piece, grid)):
                current_piece.x -= 1
        if event.key == pygame.K_DOWN:
            current_piece.y += 1
            if not(valid_space(current_piece, grid)):
                current_piece.y -= 1
        if event.key == pygame.K_UP:
            current_piece.rotation += 1
            if not(valid_space(current_piece, grid)):
                current_piece.rotation -= 1

shape_pos = convert_shape_format(current_piece)

for i in range(len(shape_pos)):
    x, y = shape_pos[i]
    if y > -1:
        grid[y][x] = current_piece.color

if change_piece:
    for pos in shape_pos:
        p = (pos[0], pos[1])
        locked_positions[p] = current_piece.color
    current_piece = next_piece
    next_piece = get_shape()
    change_piece = False
```

```
            score += clear_rows(grid, locked_positions) * 10

        draw_window(win, grid, score, last_score)
        draw_next_shape(next_piece, win)
        pygame.display.update()

        if check_lost(locked_positions):
            draw_text_middle(win, "YOU LOST!", 80, (255,255,255))
            pygame.display.update()
            pygame.time.delay(1500)
            run = False
            update_score(score)



def main_menu(win):
    run = True
    while run:
        win.fill((0,0,0))
        draw_text_middle(win, 'Press Any Key To Play', 60, (255,255,255))
        pygame.display.update()
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False
            if event.type == pygame.KEYDOWN:
                main(win)
```

```
    pygame.display.quit()


win = pygame.display.set_mode((s_width, s_height))
pygame.display.set_caption('Block Puzzel')
main_menu(win)
```

Output:



Discussion: To build this game we will be using the Pygame module in Python. We will also use the random module to select the shape and color of the Block. And we consider the whole game board and the blocks as matrices. This makes it easy to do the operations, shifting, and rotation.