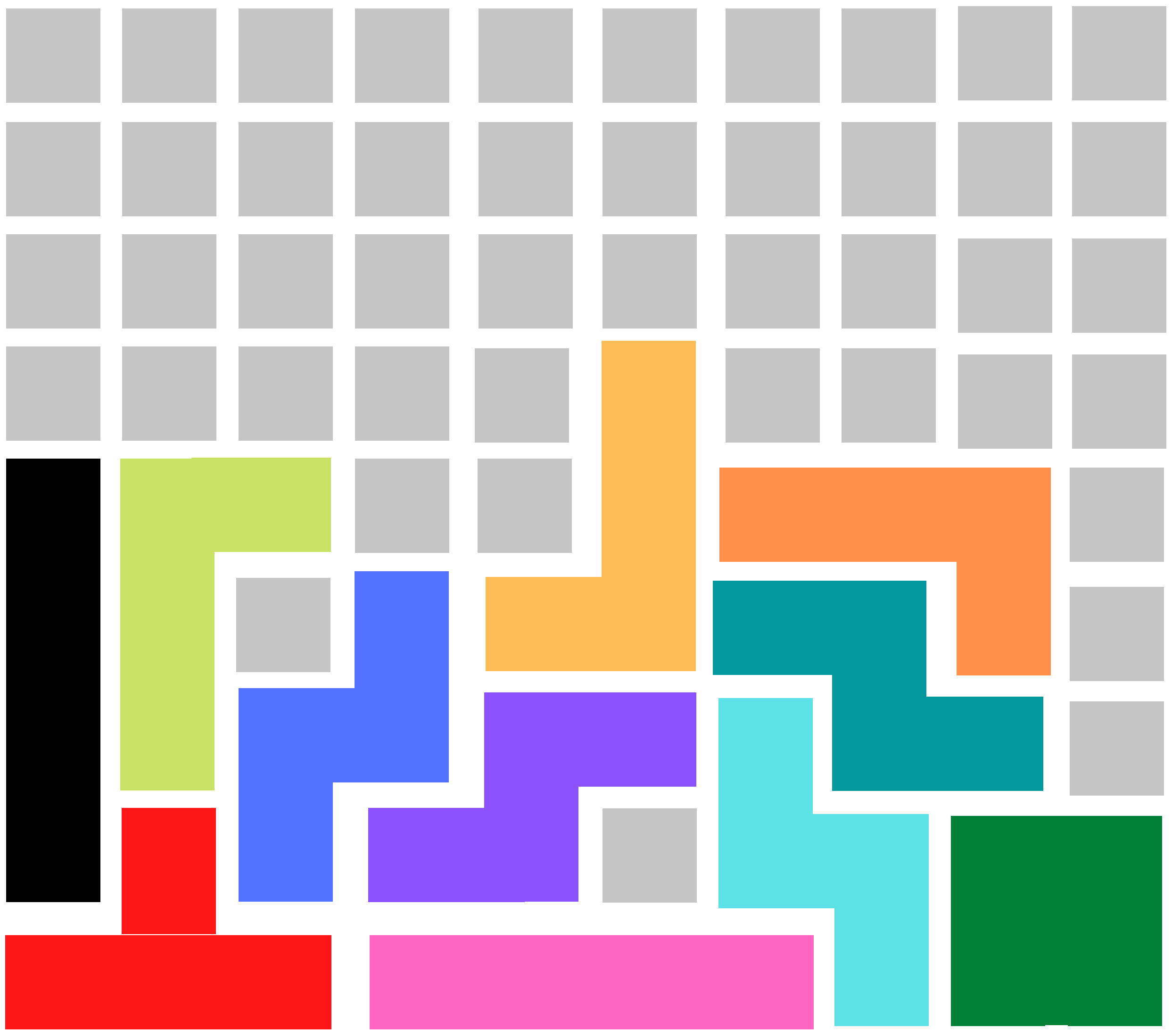
Tetris

CBSE PROJECT 2022-23
COMPUTER SCIENCE
SUBJECT CODE - 083



PIHU JHA
12 CORBETT
ROLL NO

SUBMITTED BY:

NAME: Pihu Jha

SCHOOL: Shiv Nadar School, Noida

YEAR: 2022-23

ROLL NO: __

UNDER THE GUIDANCE OF:

Ms. Manpreet Kalra

Department of Computer Science

Shiv Nadar School

Plot No SS-1 Sector 168

Expressway, Noida

Uttar Pradesh - 201305

CERTIFICATE OF AUTHENTICITY

This is to certify that Pihu Jha bearing Roll Number__ is a student of Class XII of Shiv Nadar School, Noida.

She has successfully completed her/his project under my guidance and supervision towards the fulfilment of the practical examination in Computer Science (083) conducted by the Central Board of Secondary Education for the academic year 2022 -23.

Date & Day of Submission: 4 December 2022, Sunday

Name of Subject Teachers: Ms. Manpreet Kalra
Subject Teacher's Signature:

Principal's Signature & School Stamp:

ACKNOWLEDGEMENT

I, Pihu Jha, do hereby declare that this project is my original work and I would like to thank Ms Manpreet Kalra my subject teacher, for her wholehearted support and guidance for making it possible to complete this project on time and our Principal Ms. Anju Soni for extending every possible support for the completion of this project work. I would also like to thank CBSE for giving us an opportunity to widen our knowledge base by introducing this topic of study and my school for giving us this subject option. I would also like to thank my friends and family members for their kind support and guidance without which this project could not have been completed.

Table Of Contents

Technology Used

Python:

Python is an interpreted, high level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed. Python was created in the late 1980s, and first released in 1991, by Guido van Rossum as a successor to the ABC programming language. It supports multiple libraries such as pandas, tkinter and NumPy.

Technology Used

Random Module:
This module implements pseudo-random number generators for various distributions.
For integers, there is a uniform selection from a range. For sequences, there is a uniform selection of a random element, a function to generate a random permutation of a list in place, and a function for random sampling without replacement.
On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.
Almost all module functions depend on the basic function random(), which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator.

```
random.randrange(stop)
random.randrange(start, stop[, step])
```

Technology Used

Pygame:

This is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language.

Pygame is highly portable and runs on nearly every platform and operating system.

Pygame itself has been downloaded millions of times.

Pygame is free. Released under the LGPL license, you can create open-source, freeware, shareware, and commercial games with it.

About the project

The aim of Tetris is simple; you bring down blocks from the top of the screen. You can move the blocks around, either left to right, and/or you can rotate them. The blocks fall at a certain rate, but you can make them fall faster if you're sure of your positioning. Your objective is to get all the blocks to fill all the empty space in a line at the bottom of the screen; whenever you do this, you'll find that the blocks vanish and you get awarded some points.

Tetris offers an incredibly simple reason to play—pitting your wits against the computerized block dropper in order to last as long as you can.

Tetris has very simple rules: you can only move the pieces in specific ways; your game is over if your pieces reach the top of the screen, and you can only remove pieces from the screen by filling all the blank spaces in a line.

Logic behind the code

- **Making the blocks**

The main list contains figure types, and the inner lists contain their rotations. The numbers in each figure represent the positions in a 4x4 matrix where the figure is solid. For instance, figure [1,5,9,13] represents a line.

- **The init function**

We randomly pick a type and a color

- **Initializing the game with some basic variables**

The state tells us if we are still playing a game or not. The field is the field of the game that contains zeros where it is empty, and the colors where there are figures (except the one that is still flying down).

- **The first steps of the game**

Creating a field with the size height x width.
Creating a new figure and positioning it at coordinates (3,0), which is the midpoint of the top row of the matrix

- **Checking intersections between blocks**

If the currently flying figure intersects with something fixed on the field. This may happen when the figure is moving left, right, down, or rotating.

Logic behind the code

- **Checking if we can still move the block**

We go and check each cell in the 4x4 matrix of the current Figure, whether it is out of game bounds and whether it is touching some busy game field. We check if self.field[..][..] > 0, because there may be any color. And if there is a zero, that means that the field is empty, so there is no problem.
Having this function, we can now check if we are allowed to move or rotate the figure. If it moves down and intersects, then this means we have reached the bottom, so we need to “freeze” the figure on our field

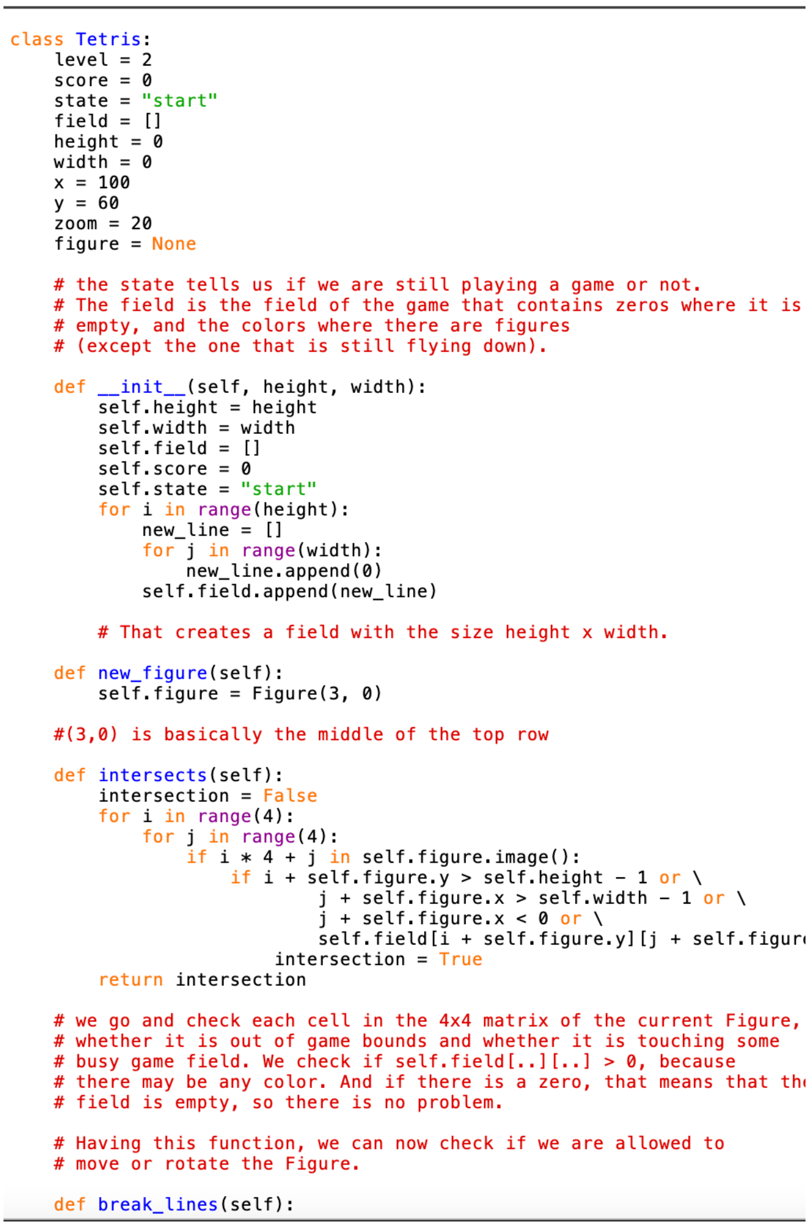
- Checking for full horizontal and vertical lines

After freezing, we have to check if there are some full horizontal lines that should be destroyed. Then we create a new Figure, and if it already intersects, then game over

Program code

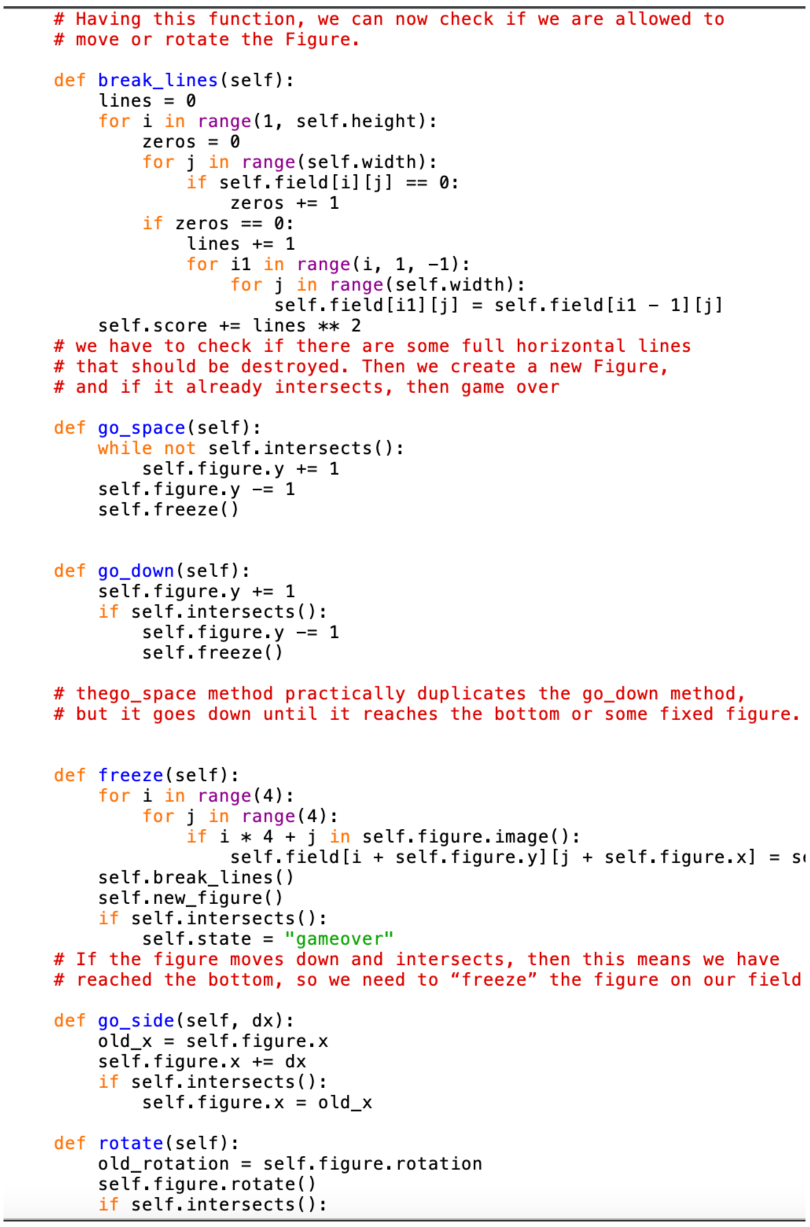
```python
import pygame
import random

colors = [
    (0, 0, 0),
    (120, 37, 179),
    (100, 179, 179),
    (80, 34, 22),
    (80, 134, 22),
    (180, 34, 22),
    (180, 34, 122),
]


class Figure:
    x = 0
    y = 0

    figures = [
        [[1, 5, 9, 13], [4, 5, 6, 7]],
        [[4, 5, 9, 10], [2, 6, 5, 9]],
        [[6, 7, 9, 10], [1, 5, 6, 10]],
        [[1, 2, 5, 9], [0, 4, 5, 6], [1, 5, 9, 8], [4, 5, 6, 10]],
        [[1, 2, 6, 10], [5, 6, 7, 9], [2, 6, 10, 11], [3, 5, 6, 7]],
        [[1, 4, 5, 6], [1, 4, 5, 9], [4, 5, 6, 9], [1, 5, 6, 9]],
        [[1, 2, 5, 6]],
        [[5, 6, 10, 11], [2, 5, 6, 9]],
        [[5, 6, 8, 9], [1, 5, 6, 10]]
    ]

    # The main list contains figure types, and the inner lists
    # contain their rotations. The numbers in each figure
    # represent the positions in a 4x4 matrix where the figure is solid.
    # For instance, the figure [1,5,9,13] represents a line.

    def __init__(self, x, y):
        self.x = x
        self.y = y
        self.type = random.randint(0, len(self.figures) - 1)
        self.color = random.randint(1, len(colors) - 1)
        self.rotation = 0

    #we randomly pick a type and a color.

    def image(self):
        return self.figures[self.type][self.rotation]

    def rotate(self):
        self.rotation = (self.rotation + 1) % len(self.figures[self.type


class Tetris:
    level = 2
    score = 0
    state = "start"
    field = []
    height = 0
    width = 0
    x = 100
```

Program code



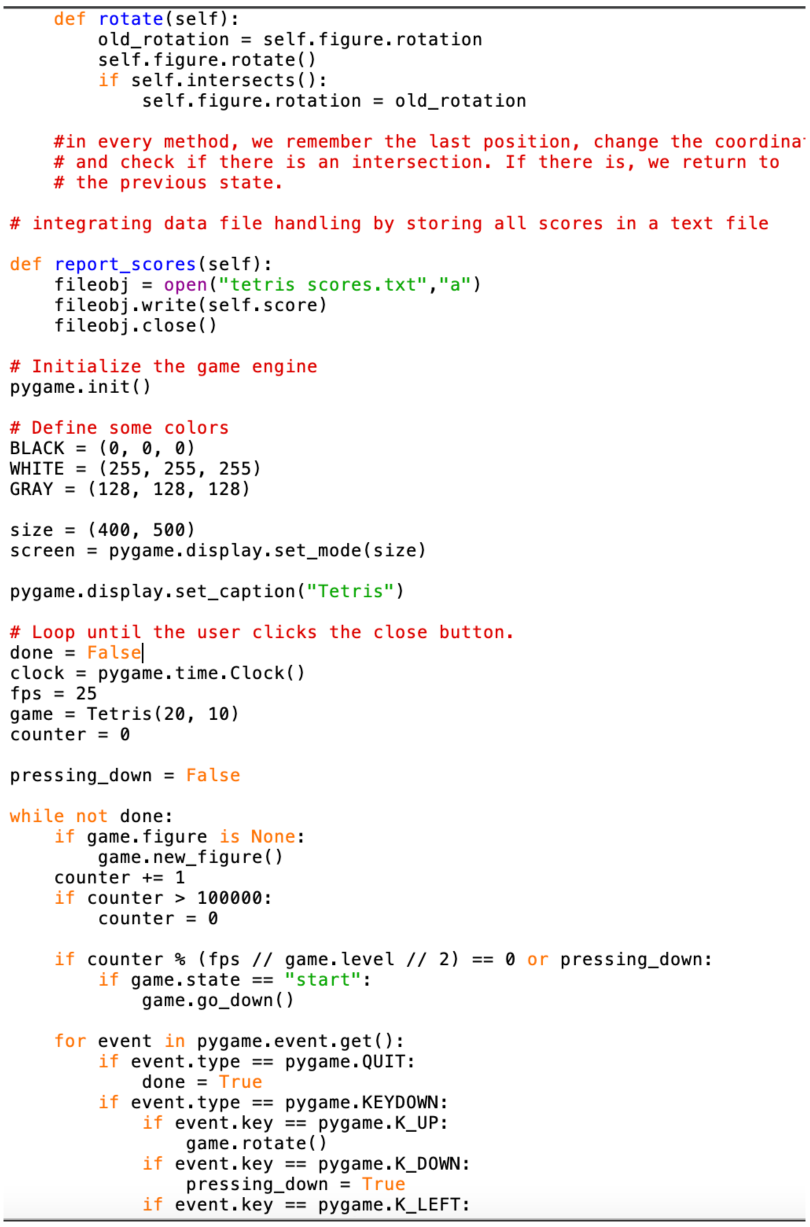
```
class Tetris:
    level = 2
    score = 0
    state = "start"
    field = []
    height = 0
    width = 0
    x = 100
    y = 60
    zoom = 20
    figure = None

    # the state tells us if we are still playing a game or not.
    # The field is the field of the game that contains zeros where it is
    # empty, and the colors where there are figures
    # (except the one that is still flying down).

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

        # That creates a field with the size height x width.

    def new_figure(self):
        self.figure = Figure(3, 0)

    #(3,0) is basically the middle of the top row

    def intersects(self):
        intersection = False
        for i in range(4):
            for j in range(4):
                if i * 4 + j in self.figure.image():
                    if i + self.figure.y > self.height - 1 or \
                            j + self.figure.x > self.width - 1 or \
                            j + self.figure.x < 0 or \
                            self.field[i + self.figure.y][j + self.figur
                        intersection = True
        return intersection

    # we go and check each cell in the 4x4 matrix of the current Figure,
    # whether it is out of game bounds and whether it is touching some
    # busy game field. We check if self.field[..][..] > 0, because
    # there may be any color. And if there is a zero, that means that th
    # field is empty, so there is no problem.

    # Having this function, we can now check if we are allowed to
    # move or rotate the Figure.

    def break_lines(self):
```

Program code



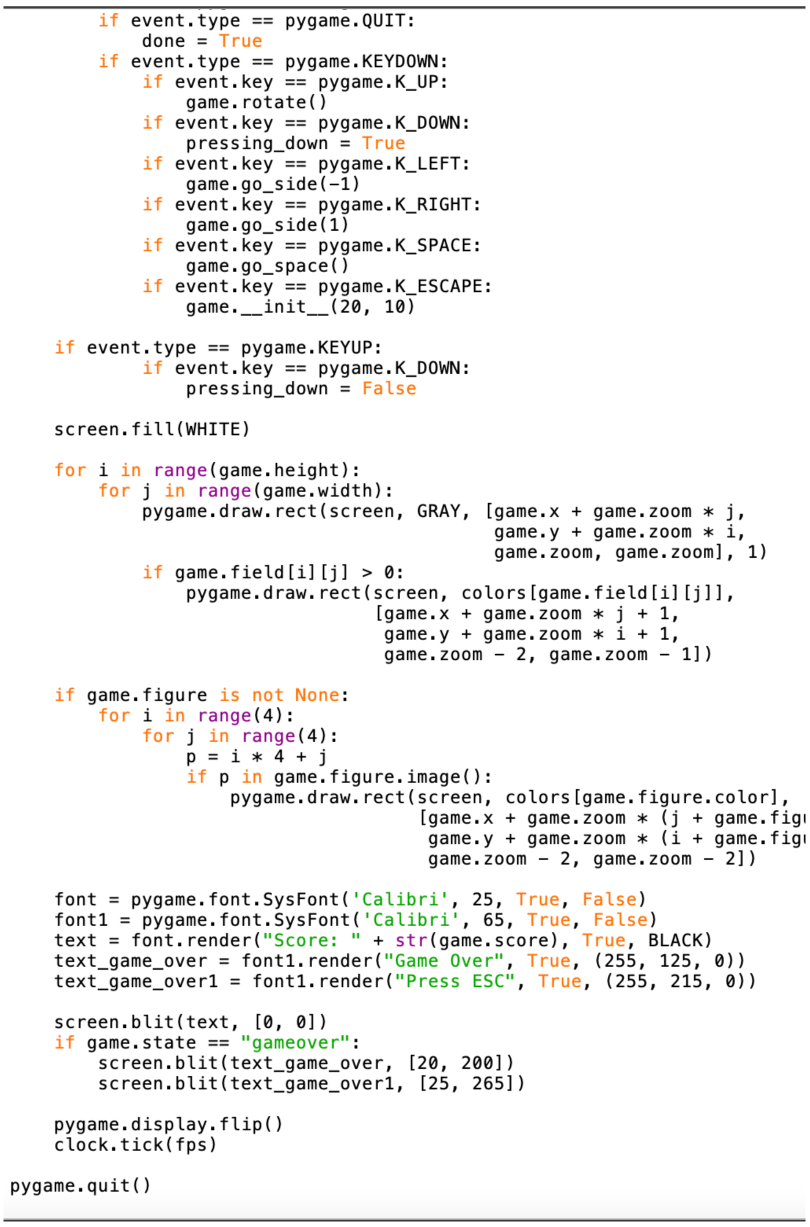
```
# Having this function, we can now check if we are allowed to
# move or rotate the Figure.

def break_lines(self):
    lines = 0
    for i in range(1, self.height):
        zeros = 0
        for j in range(self.width):
            if self.field[i][j] == 0:
                zeros += 1
        if zeros == 0:
            lines += 1
            for i1 in range(i, 1, -1):
                for j in range(self.width):
                    self.field[i1][j] = self.field[i1 - 1][j]
    self.score += lines ** 2
# we have to check if there are some full horizontal lines
# that should be destroyed. Then we create a new Figure,
# and if it already intersects, then game over

def go_space(self):
    while not self.intersects():
        self.figure.y += 1
    self.figure.y -= 1
    self.freeze()


def go_down(self):
    self.figure.y += 1
    if self.intersects():
        self.figure.y -= 1
        self.freeze()

# thego_space method practically duplicates the go_down method,
# but it goes down until it reaches the bottom or some fixed figure.


def freeze(self):
    for i in range(4):
        for j in range(4):
            if i * 4 + j in self.figure.image():
                self.field[i + self.figure.y][j + self.figure.x] = s
    self.break_lines()
    self.new_figure()
    if self.intersects():
        self.state = "gameover"
# If the figure moves down and intersects, then this means we have
# reached the bottom, so we need to “freeze” the figure on our field

def go_side(self, dx):
    old_x = self.figure.x
    self.figure.x += dx
    if self.intersects():
        self.figure.x = old_x

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

Program code



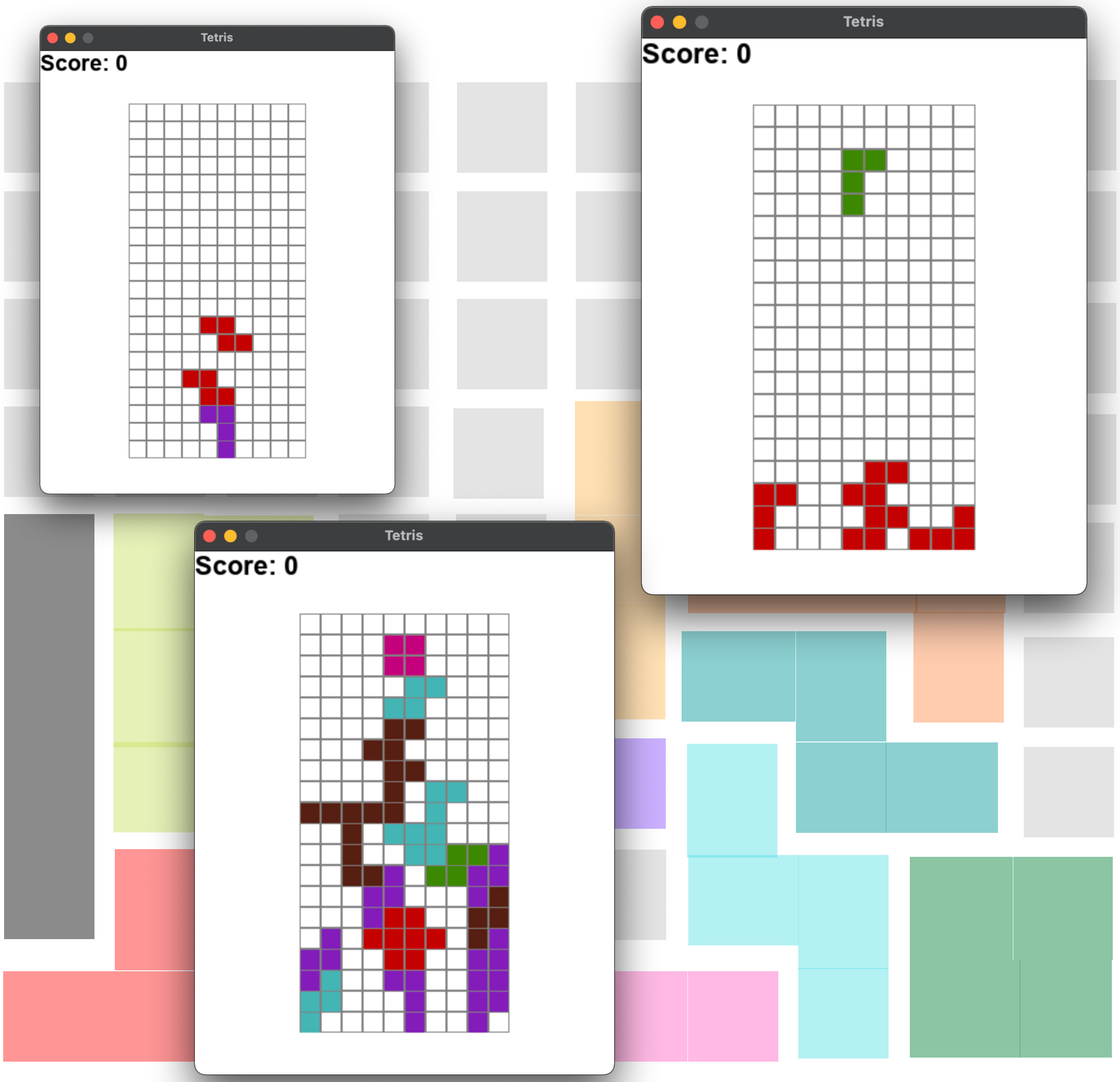
```
    def rotate(self):
        old_rotation = self.figure.rotation
        self.figure.rotate()
        if self.intersects():
            self.figure.rotation = old_rotation

    #in every method, we remember the last position, change the coordina
    # and check if there is an intersection. If there is, we return to
    # the previous state.

# integrating data file handling by storing all scores in a text file

def report_scores(self):
    fileobj = open("tetris scores.txt","a")
    fileobj.write(self.score)
    fileobj.close()

# Initialize the game engine
pygame.init()

# Define some colors
BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GRAY = (128, 128, 128)

size = (400, 500)
screen = pygame.display.set_mode(size)

pygame.display.set_caption("Tetris")

# Loop until the user clicks the close button.
done = False
clock = pygame.time.Clock()
fps = 25
game = Tetris(20, 10)
counter = 0

pressing_down = False

while not done:
    if game.figure is None:
        game.new_figure()
    counter += 1
    if counter > 100000:
        counter = 0

    if counter % (fps // game.level // 2) == 0 or pressing_down:
        if game.state == "start":
            game.go_down()

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                game.rotate()
            if event.key == pygame.K_DOWN:
                pressing_down = True
            if event.key == pygame.K_LEFT:
```

Program code



```
            if event.type == pygame.QUIT:
                done = True
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_UP:
                    game.rotate()
                if event.key == pygame.K_DOWN:
                    pressing_down = True
                if event.key == pygame.K_LEFT:
                    game.go_side(-1)
                if event.key == pygame.K_RIGHT:
                    game.go_side(1)
                if event.key == pygame.K_SPACE:
                    game.go_space()
                if event.key == pygame.K_ESCAPE:
                    game.__init__(20, 10)

    if event.type == pygame.KEYUP:
                if event.key == pygame.K_DOWN:
                    pressing_down = False

    screen.fill(WHITE)

    for i in range(game.height):
        for j in range(game.width):
            pygame.draw.rect(screen, GRAY, [game.x + game.zoom * j,
                                            game.y + game.zoom * i,
                                            game.zoom, game.zoom], 1)
            if game.field[i][j] > 0:
                pygame.draw.rect(screen, colors[game.field[i][j]],
                                 [game.x + game.zoom * j + 1,
                                  game.y + game.zoom * i + 1,
                                  game.zoom - 2, game.zoom - 1])

    if game.figure is not None:
        for i in range(4):
            for j in range(4):
                p = i * 4 + j
                if p in game.figure.image():
                    pygame.draw.rect(screen, colors[game.figure.color],
                                     [game.x + game.zoom * (j + game.fig
                                      game.y + game.zoom * (i + game.fig
                                      game.zoom - 2, game.zoom - 2])

    font = pygame.font.SysFont('Calibri', 25, True, False)
    font1 = pygame.font.SysFont('Calibri', 65, True, False)
    text = font.render("Score: " + str(game.score), True, BLACK)
    text_game_over = font1.render("Game Over", True, (255, 125, 0))
    text_game_over1 = font1.render("Press ESC", True, (255, 215, 0))

    screen.blit(text, [0, 0])
    if game.state == "gameover":
        screen.blit(text_game_over, [20, 200])
        screen.blit(text_game_over1, [25, 265])

    pygame.display.flip()
    clock.tick(fps)

pygame.quit()
```

Output



Bibliography

- Grade 12 Computer Science NCERT
- Python Random Module: https://docs.python.org/3/library/random.html
- Pygame: https://www.pygame.org/wiki/about
- Geeks for Geeks: https://www.geeksforgeeks.org/reading-writing-text-files-python/
- W3 Schools: https://www.w3schools.com/python/
- A Game Explained: https://www.interaction-design.org/literature/article/a-game-explained-an-example-of-a-single-game-and-how-it-meets-the-rules-of-fun#:~:text=The%20aim%20in%20Tetris%20is,re%20sure%20of%20your%20positioning.