**BLOCKS**

from colors import Colors

import pygame

from position import Position

class Block:

    def \_\_init\_\_(self, id):

        self.id = id

        self.cells = {}

        self.cell\_size = 30

        self.row\_offset = 0

        self.column\_offset = 0

        self.rotation\_state = 0

        self.colors = Colors.get\_cell\_colors()

    def move(self, rows, columns):

        self.row\_offset += rows

        self.column\_offset += columns

    def get\_cell\_positions(self):

        tiles = self.cells[self.rotation\_state]

        moved\_tiles = []

        for position in tiles:

            position = Position(position.row + self.row\_offset, position.column + self.column\_offset)

            moved\_tiles.append(position)

        return moved\_tiles

    def rotate(self):

        self.rotation\_state += 1

        if self.rotation\_state == len(self.cells):

            self.rotation\_state = 0

    def undo\_rotation(self):

        self.rotation\_state -= 1

        if self.rotation\_state == -1:

            self.rotation\_state = len(self.cells) - 1

    def draw(self, screen, offset\_x, offset\_y):

        tiles = self.get\_cell\_positions()

        for tile in tiles:

            tile\_rect = pygame.Rect(offset\_x + tile.column \* self.cell\_size,

                offset\_y + tile.row \* self.cell\_size, self.cell\_size -1, self.cell\_size -1)

            pygame.draw.rect(screen, self.colors[self.id], tile\_rect)

**COLOR**

class Colors:

    dark\_grey = (26, 31, 40)

    green = (47, 230, 23)

    red = (232, 18, 18)

    orange = (226, 116, 17)

    yellow = (237, 234, 4)

    purple = (166, 0, 247)

    cyan = (21, 204, 209)

    blue = (13, 64, 216)

    white = (255, 255, 255)

    dark\_blue = (44, 44, 127)

    light\_blue = (59, 85, 162)

    @classmethod

    def get\_cell\_colors(cls):

        return [cls.dark\_grey, cls.green, cls.red, cls.orange, cls.yellow, cls.purple, cls.cyan, cls.blue]

**GAME**

from grid import Grid

from blocks import \*

import random

import pygame

class Game:

    def \_\_init\_\_(self):

        self.grid = Grid()

        self.blocks = [IBlock(), JBlock(), LBlock(), OBlock(), SBlock(), TBlock(), ZBlock()]

        self.current\_block = self.get\_random\_block()

        self.next\_block = self.get\_random\_block()

        self.game\_over = False

        self.score = 0

        self.rotate\_sound = pygame.mixer.Sound("Sounds/rotate.ogg")

        self.clear\_sound = pygame.mixer.Sound("Sounds/clear.ogg")

        pygame.mixer.music.load("Sounds/music.ogg")

        pygame.mixer.music.play(-1)

    def update\_score(self, lines\_cleared, move\_down\_points):

        if lines\_cleared == 1:

            self.score += 100

        elif lines\_cleared == 2:

            self.score += 300

        elif lines\_cleared == 3:

            self.score += 500

        self.score += move\_down\_points

    def get\_random\_block(self):

        if len(self.blocks) == 0:

            self.blocks = [IBlock(), JBlock(), LBlock(), OBlock(), SBlock(), TBlock(), ZBlock()]

        block = random.choice(self.blocks)

        self.blocks.remove(block)

        return block

    def move\_left(self):

        self.current\_block.move(0, -1)

        if self.block\_inside() == False or self.block\_fits() == False:

            self.current\_block.move(0, 1)

    def move\_right(self):

        self.current\_block.move(0, 1)

        if self.block\_inside() == False or self.block\_fits() == False:

            self.current\_block.move(0, -1)

    def move\_down(self):

        self.current\_block.move(1, 0)

        if self.block\_inside() == False or self.block\_fits() == False:

            self.current\_block.move(-1, 0)

            self.lock\_block()

    def lock\_block(self):

        tiles = self.current\_block.get\_cell\_positions()

        for position in tiles:

            self.grid.grid[position.row][position.column] = self.current\_block.id

        self.current\_block = self.next\_block

        self.next\_block = self.get\_random\_block()

        rows\_cleared = self.grid.clear\_full\_rows()

        if rows\_cleared > 0:

            self.clear\_sound.play()

            self.update\_score(rows\_cleared, 0)

        if self.block\_fits() == False:

            self.game\_over = True

    def reset(self):

        self.grid.reset()

        self.blocks = [IBlock(), JBlock(), LBlock(), OBlock(), SBlock(), TBlock(), ZBlock()]

        self.current\_block = self.get\_random\_block()

        self.next\_block = self.get\_random\_block()

        self.score = 0

    def block\_fits(self):

        tiles = self.current\_block.get\_cell\_positions()

        for tile in tiles:

            if self.grid.is\_empty(tile.row, tile.column) == False:

                return False

        return True

    def rotate(self):

        self.current\_block.rotate()

        if self.block\_inside() == False or self.block\_fits() == False:

            self.current\_block.undo\_rotation()

        else:

            self.rotate\_sound.play()

    def block\_inside(self):

        tiles = self.current\_block.get\_cell\_positions()

        for tile in tiles:

            if self.grid.is\_inside(tile.row, tile.column) == False:

                return False

        return True

    def draw(self, screen):

        self.grid.draw(screen)

        self.current\_block.draw(screen, 11, 11)

        if self.next\_block.id == 3:

            self.next\_block.draw(screen, 255, 290)

        elif self.next\_block.id == 4:

            self.next\_block.draw(screen, 255, 280)

        else:

            self.next\_block.draw(screen, 270, 270)

**GRID.PY**

import pygame

from colors import Colors

class Grid:

    def \_\_init\_\_(self):

        self.num\_rows = 20

        self.num\_cols = 10

        self.cell\_size = 30

        self.grid = [[0 for j in range(self.num\_cols)] for i in range(self.num\_rows)]

        self.colors = Colors.get\_cell\_colors()

    def print\_grid(self):

        for row in range(self.num\_rows):

            for column in range(self.num\_cols):

                print(self.grid[row][column], end = " ")

            print()

    def is\_inside(self, row, column):

        if row >= 0 and row < self.num\_rows and column >= 0 and column < self.num\_cols:

            return True

        return False

    def is\_empty(self, row, column):

        if self.grid[row][column] == 0:

            return True

        return False

    def is\_row\_full(self, row):

        for column in range(self.num\_cols):

            if self.grid[row][column] == 0:

                return False

        return True

    def clear\_row(self, row):

        for column in range(self.num\_cols):

            self.grid[row][column] = 0

    def move\_row\_down(self, row, num\_rows):

        for column in range(self.num\_cols):

            self.grid[row+num\_rows][column] = self.grid[row][column]

            self.grid[row][column] = 0

    def clear\_full\_rows(self):

        completed = 0

        for row in range(self.num\_rows-1, 0, -1):

            if self.is\_row\_full(row):

                self.clear\_row(row)

                completed += 1

            elif completed > 0:

                self.move\_row\_down(row, completed)

        return completed

    def reset(self):

        for row in range(self.num\_rows):

            for column in range(self.num\_cols):

                self.grid[row][column] = 0

    def draw(self, screen):

        for row in range(self.num\_rows):

            for column in range(self.num\_cols):

                cell\_value = self.grid[row][column]

                cell\_rect = pygame.Rect(column\*self.cell\_size + 11, row\*self.cell\_size + 11,

                self.cell\_size -1, self.cell\_size -1)

                pygame.draw.rect(screen, self.colors[cell\_value], cell\_rect**)**

**MAIN.PY**

import pygame,sys

from game import Game

from colors import Colors

pygame.init()

title\_font = pygame.font.Font(None, 40)

score\_surface = title\_font.render("Score", True, Colors.white)

next\_surface = title\_font.render("Next", True, Colors.white)

game\_over\_surface = title\_font.render("GAME OVER", True, Colors.white)

score\_rect = pygame.Rect(320, 55, 170, 60)

next\_rect = pygame.Rect(320, 215, 170, 180)

screen = pygame.display.set\_mode((500, 620))

pygame.display.set\_caption("Python Tetris")

clock = pygame.time.Clock()

game = Game()

GAME\_UPDATE = pygame.USEREVENT

pygame.time.set\_timer(GAME\_UPDATE, 200)

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

if event.type == pygame.KEYDOWN:

if game.game\_over == True:

game.game\_over = False

game.reset()

if event.key == pygame.K\_LEFT and game.game\_over == False:

game.move\_left()

if event.key == pygame.K\_RIGHT and game.game\_over == False:

game.move\_right()

if event.key == pygame.K\_DOWN and game.game\_over == False:

game.move\_down()

game.update\_score(0, 1)

if event.key == pygame.K\_UP and game.game\_over == False:

game.rotate()

if event.type == GAME\_UPDATE and game.game\_over == False:

game.move\_down()

#Drawing

score\_value\_surface = title\_font.render(str(game.score), True, Colors.white)

screen.fill(Colors.dark\_blue)

screen.blit(score\_surface, (365, 20, 50, 50))

screen.blit(next\_surface, (375, 180, 50, 50))

if game.game\_over == True:

screen.blit(game\_over\_surface, (320, 450, 50, 50))

pygame.draw.rect(screen, Colors.light\_blue, score\_rect, 0, 10)

screen.blit(score\_value\_surface, score\_value\_surface.get\_rect(centerx = score\_rect.centerx,

centery = score\_rect.centery))

pygame.draw.rect(screen, Colors.light\_blue, next\_rect, 0, 10)

game.draw(screen)

pygame.display.update()

clock.tick(60)

**POSITION.PY**

class Position:

    def \_\_init\_\_(self, row, column):

        self.row = row

        self.column = column