```
"""
SWEET MATCH  -  a Candy Crush style match-3 game (Python + Pygame)

How to play:
  * Tap a candy, then tap a neighbour candy to swap them.   (or swipe a candy)
  * Make a line of 3 or more same candies to pop them.
  * Reach the goal score before your moves run out!

Run on a laptop (VS Code / Visual Studio / PyCharm):
      pip install pygame
      python sweet_match.py
Also works in browser pygame editors.
"""
import asyncio
import math
import random
import pygame

W, H = 360, 640
ROWS = COLS = 8
CELL = 40
BX, BY = 20, 190                      # top-left corner of the board
BOARD_W, BOARD_H = COLS * CELL, ROWS * CELL
SWAP_SPEED = 330                      # pixels per second
GRAVITY = 2600
POP_TIME = 0.25

COLORS = [(235, 60, 70), (255, 150, 40), (255, 215, 40),
          (70, 200, 90), (60, 140, 255), (170, 80, 220)]
PRAISE = {2: 'Sweet!', 3: 'Tasty!', 4: 'Delicious!', 5: 'Divine!'}

pygame.init()
screen = pygame.display.set_mode((W, H))
pygame.display.set_caption('Sweet Match')
clock = pygame.time.Clock()
f_small = pygame.font.Font(None, 22)
f_mid = pygame.font.Font(None, 30)
f_big = pygame.font.Font(None, 48)
f_title = pygame.font.Font(None, 54)

g = {}                                # the whole game state lives here


# ========================================================== background art
def make_background():
    bg = pygame.Surface((W, H))
    for y in range(H):                # pink -> purple gradient
        k = y / H
        color = (int(255 - 115 * k), int(150 - 60 * k), int(200 + 20 * k))
        pygame.draw.line(bg, color, (0, y), (W, y))
    bubbles = pygame.Surface((W, H), pygame.SRCALPHA)
```

```
    rnd = random.Random(4)
    for _ in range(14):
        pygame.draw.circle(bubbles, (255, 255, 255, 35),
                           (rnd.randint(0, W), rnd.randint(0, H)), rnd.randint(15, 50))
    bg.blit(bubbles, (0, 0))
    return bg


def make_board_surface():
    s = pygame.Surface((W, H), pygame.SRCALPHA)
    pygame.draw.rect(s, (255, 255, 255, 70), (BX - 8, BY - 8, BOARD_W + 16, BOARD_H + 16),
                     border_radius=18)
    for r in range(ROWS):
        for c in range(COLS):
            a = 55 if (r + c) % 2 == 0 else 25
            pygame.draw.rect(s, (255, 255, 255, a), (BX + c * CELL, BY + r * CELL, CELL, CELL))
    return s


BACKGROUND = make_background()
BOARD_SURFACE = make_board_surface()


# ============================================================ small helpers
def text(msg, font, color, x, y, shadow=True):
    if shadow:
        sh = font.render(msg, True, (60, 20, 90))
        screen.blit(sh, sh.get_rect(center=(x + 2, y + 2)))
    img = font.render(msg, True, color)
    screen.blit(img, img.get_rect(center=(x, y)))


def cell_center(r, c):
    return BX + c * CELL + CELL // 2, BY + r * CELL + CELL // 2


def cell_at(pos):
    c, r = (pos[0] - BX) // CELL, (pos[1] - BY) // CELL
    if 0 <= r < ROWS and 0 <= c < COLS:
        return int(r), int(c)
    return None


def kind(r, c):
    candy = g['grid'][r][c]
    return candy['t'] if candy else None


def make_candy(t, oy=0.0):
    return {'t': t, 'ox': 0.0, 'oy': oy, 'vy': 0.0, 'mode': 'fall', 's': 1.0}


def star_points(cx, cy, R, n=5):
    pts = []
    for i in range(n * 2):
```

```
        a = -math.pi / 2 + i * math.pi / n
        rad = R if i % 2 == 0 else R * 0.5
        pts.append((cx + math.cos(a) * rad, cy + math.sin(a) * rad))
    return pts


# ========================================================== drawing candies
def draw_candy(t, cx, cy, scale=1.0):
    if scale <= 0.02:
        return
    r = 16 * scale
    color = COLORS[t]
    dark = tuple(int(v * 0.55) for v in color)
    light = tuple(int(v + (255 - v) * 0.65) for v in color)

    def shape(col, grow):
        rr = r + grow
        if t == 0:                                          # round candy
            pygame.draw.circle(screen, col, (cx, cy), rr)
        elif t == 1:                                        # rounded square
            pygame.draw.rect(screen, col, (cx - rr, cy - rr, rr * 2, rr * 2),
                             border_radius=max(2, int(rr * 0.4)))
        elif t == 2:                                        # star
            pygame.draw.polygon(screen, col, star_points(cx, cy + 1, rr * 1.25))
        elif t == 3:                                        # diamond
            pygame.draw.polygon(screen, col, [(cx, cy - rr * 1.15), (cx + rr * 1.05, cy),
                                              (cx, cy + rr * 1.15), (cx - rr * 1.05, cy)])
        elif t == 4:                                        # hexagon
            pygame.draw.polygon(screen, col, [(cx + math.cos(i * math.pi / 3) * rr * 1.1,
                                               cy + math.sin(i * math.pi / 3) * rr * 1.1)
                                              for i in range(6)])
        else:                                               # heart
            pygame.draw.circle(screen, col, (cx - rr * 0.5, cy - rr * 0.2), rr * 0.58)
            pygame.draw.circle(screen, col, (cx + rr * 0.5, cy - rr * 0.2), rr * 0.58)
            pygame.draw.polygon(screen, col, [(cx - rr * 1.04, cy - rr * 0.05),
                                              (cx + rr * 1.04, cy - rr * 0.05),
                                              (cx, cy + rr * 1.05)])

    shape(dark, 2.5 * scale)
    shape(color, 0)
    pygame.draw.ellipse(screen, light, (cx - r * 0.55, cy - r * 0.65, r * 0.55, r * 0.38))


def draw_stars(cx, cy, earned):
    for i in range(3):
        x = cx + (i - 1) * 48
        col = (255, 215, 40) if i < earned else (110, 90, 130)
        pygame.draw.polygon(screen, (90, 60, 20), star_points(x, cy, 24))
        pygame.draw.polygon(screen, col, star_points(x, cy, 20))


# ========================================================== game logic
def find_matches():
    cells, runs = set(), []
    for r in range(ROWS):                                   # rows
```

```
        c = 0
        while c < COLS:
            t = kind(r, c)
            if t is None:
                c += 1
                continue
            k = c
            while k + 1 < COLS and kind(r, k + 1) == t:
                k += 1
            if k - c + 1 >= 3:
                run = [(r, x) for x in range(c, k + 1)]
                runs.append(run)
                cells.update(run)
            c = k + 1
    for c in range(COLS):                                   # columns
        r = 0
        while r < ROWS:
            t = kind(r, c)
            if t is None:
                r += 1
                continue
            k = r
            while k + 1 < ROWS and kind(k + 1, c) == t:
                k += 1
            if k - r + 1 >= 3:
                run = [(x, c) for x in range(r, k + 1)]
                runs.append(run)
                cells.update(run)
            r = k + 1
    return (cells, runs) if cells else None


def swap_cells(a, b):
    grid = g['grid']
    grid[a[0]][a[1]], grid[b[0]][b[1]] = grid[b[0]][b[1]], grid[a[0]][a[1]]


def find_move():
    for r in range(ROWS):
        for c in range(COLS):
            for dr, dc in ((0, 1), (1, 0)):
                r2, c2 = r + dr, c + dc
                if r2 < ROWS and c2 < COLS:
                    swap_cells((r, c), (r2, c2))
                    found = find_matches()
                    swap_cells((r, c), (r2, c2))
                    if found:
                        return (r, c), (r2, c2)
    return None


def new_board():
    n = g['ntypes']
    while True:
        grid = [[None] * COLS for _ in range(ROWS)]
```

```
        for r in range(ROWS):
            for c in range(COLS):
                ok = []
                for t in range(n):
                    if c >= 2 and grid[r][c - 1]['t'] == t and grid[r][c - 2]['t'] == t:
                        continue
                    if r >= 2 and grid[r - 1][c]['t'] == t and grid[r - 2][c]['t'] == t:
                        continue
                    ok.append(t)
                grid[r][c] = make_candy(random.choice(ok), -(BOARD_H + 40))   # drops in
        g['grid'] = grid
        if find_move():
            break
    g['phase'], g['after'] = 'moving', 'idle'


def setup_level(level):
    g.update(level=level, target=500 + 300 * (level - 1), moves=20, score=0, combo=0,
             ntypes=5 if level < 3 else 6, selected=None, down=None, hint=None,
             popping=set(), particles=[], floaters=[], popup=None, idle_t=0.0, end_t=0.0)
    new_board()


def try_swap(a, b):
    if g['phase'] != 'idle':
        return
    swap_cells(a, b)
    for here, other in ((a, b), (b, a)):         # candy at "here" slides in from "other"
        candy = g['grid'][here[0]][here[1]]
        candy['ox'] = (other[1] - here[1]) * CELL
        candy['oy'] = (other[0] - here[0]) * CELL
        candy['mode'] = 'lin'
    g.update(phase='moving', after='check_swap', swap=(a, b), selected=None, down=None, hint=None)


def check_swap():
    found = find_matches()
    if found:
        g['moves'] -= 1
        g['combo'] = 1
        start_pop(found)
    else:                                        # bad move: swap back
        a, b = g['swap']
        try_swap_back(a, b)


def try_swap_back(a, b):
    swap_cells(a, b)
    for here, other in ((a, b), (b, a)):
        candy = g['grid'][here[0]][here[1]]
        candy['ox'] = (other[1] - here[1]) * CELL
        candy['oy'] = (other[0] - here[0]) * CELL
        candy['mode'] = 'lin'
    g.update(phase='moving', after='idle')
    g['idle_t'] = 0.0
```

```
def start_pop(found):
    cells, runs = found
    points = 10 * len(cells) * g['combo']
    for run in runs:
        points += 30 if len(run) == 4 else 80 if len(run) >= 5 else 0
    g['score'] += points
    mx = sum(cell_center(r, c)[0] for r, c in cells) / len(cells)
    my = sum(cell_center(r, c)[1] for r, c in cells) / len(cells)
    g['floaters'].append({'text': f'+{points}', 'x': mx, 'y': my, 'life': 0.9})
    if g['combo'] >= 2:
        g['popup'] = [PRAISE[min(g['combo'], 5)], 1.0]
    g.update(phase='pop', popping=set(cells), pop_t=POP_TIME)


def finish_pop():
    grid = g['grid']
    for r, c in g['popping']:
        x, y = cell_center(r, c)
        color = COLORS[grid[r][c]['t']]
        for _ in range(6):
            a = random.uniform(0, math.pi * 2)
            sp = random.uniform(60, 180)
            g['particles'].append({'x': x, 'y': y, 'vx': math.cos(a) * sp, 'vy': math.sin(a) * sp
- 60,
                                   'life': 0.5, 'color': color})
        grid[r][c] = None
    g['popping'] = set()

    for c in range(COLS):                          # gravity: candies fall down
        write = ROWS - 1
        for r in range(ROWS - 1, -1, -1):
            candy = grid[r][c]
            if candy is not None:
                if r != write:
                    grid[write][c], grid[r][c] = candy, None
                    candy['oy'] -= (write - r) * CELL
                    candy['mode'], candy['vy'] = 'fall', 0.0
                write -= 1
        missing = write + 1                        # new candies drop from above
        for r in range(missing):
            grid[r][c] = make_candy(random.randrange(g['ntypes']), -missing * CELL)
    g.update(phase='moving', after='cascade')


def cascade():
    found = find_matches()
    if found:
        g['combo'] += 1
        start_pop(found)
    else:
        end_turn()
```

```
def end_turn():
    g['combo'] = 0
    g['phase'], g['idle_t'] = 'idle', 0.0
    if g['score'] >= g['target']:
        g.update(phase='win', end_t=0.0)
        g['best'] = max(g.get('best', 0), g['score'])
    elif g['moves'] <= 0:
        g.update(phase='lose', end_t=0.0)
        g['best'] = max(g.get('best', 0), g['score'])
    elif find_move() is None:                    # no moves left: shuffle
        g['popup'] = ['No moves! Shuffling', 1.2]
        new_board()


AFTER = {'check_swap': check_swap, 'cascade': cascade,
         'idle': lambda: g.update(phase='idle', idle_t=0.0)}


def animate(dt):
    moving = False
    for row in g['grid']:
        for candy in row:
            if candy is None:
                continue
            if candy['mode'] == 'lin':
                dist = math.hypot(candy['ox'], candy['oy'])
                step = SWAP_SPEED * dt
                if dist <= step:
                    candy['ox'] = candy['oy'] = 0.0
                else:
                    k = (dist - step) / dist
                    candy['ox'] *= k
                    candy['oy'] *= k
                    moving = True
            elif candy['oy'] < 0:
                candy['vy'] += GRAVITY * dt
                candy['oy'] += candy['vy'] * dt
                if candy['oy'] >= 0:
                    candy['oy'], candy['vy'] = 0.0, 0.0
                else:
                    moving = True
    return moving


def update(dt):
    phase = g['phase']
    if phase == 'moving':
        if not animate(dt):
            AFTER[g['after']]()
    elif phase == 'pop':
        g['pop_t'] -= dt
        for r, c in g['popping']:
            g['grid'][r][c]['s'] = max(0.0, g['pop_t'] / POP_TIME)
        if g['pop_t'] <= 0:
            finish_pop()
```

```
    elif phase == 'idle':
        g['idle_t'] += dt
        if g['idle_t'] > 6 and g['hint'] is None:
            g['hint'] = find_move()
    else:
        g['end_t'] += dt

    for p in g['particles'][:]:
        p['x'] += p['vx'] * dt
        p['y'] += p['vy'] * dt
        p['vy'] += 600 * dt
        p['life'] -= dt
        if p['life'] <= 0:
            g['particles'].remove(p)
    for fl in g['floaters'][:]:
        fl['y'] -= 40 * dt
        fl['life'] -= dt
        if fl['life'] <= 0:
            g['floaters'].remove(fl)
    if g['popup']:
        g['popup'][1] -= dt
        if g['popup'][1] <= 0:
            g['popup'] = None


# ============================================================ drawing the screen
def panel(x, y, w, h, label, value):
    pygame.draw.rect(screen, (90, 40, 140), (x, y, w, h), border_radius=12)
    pygame.draw.rect(screen, (255, 255, 255), (x, y, w, h), 2, border_radius=12)
    text(label, f_small, (255, 220, 120), x + w // 2, y + 14, shadow=False)
    text(str(value), f_mid, (255, 255, 255), x + w // 2, y + 36, shadow=False)


def draw():
    screen.blit(BACKGROUND, (0, 0))
    text('SWEET MATCH', f_title, (255, 255, 255), W // 2, 34)

    panel(20, 66, 100, 52, 'SCORE', g['score'])
    panel(130, 66, 100, 52, 'LEVEL', g['level'])
    panel(240, 66, 100, 52, 'MOVES', g['moves'])

    # goal bar
    pygame.draw.rect(screen, (90, 40, 140), (20, 132, 320, 24), border_radius=12)
    fill = min(g['score'] / g['target'], 1.0)
    if fill > 0:
        pygame.draw.rect(screen, (255, 215, 40), (22, 134, max(10, int(316 * fill)), 20),
border_radius=10)
    text(f"Goal: {g['target']}", f_small, (255, 255, 255), W // 2, 144, shadow=False)

    screen.blit(BOARD_SURFACE, (0, 0))
    screen.set_clip(pygame.Rect(BX, BY, BOARD_W, BOARD_H))

    t_ms = pygame.time.get_ticks() / 1000
    hint = g['hint'] if g['phase'] == 'idle' else None
    for r in range(ROWS):
```

```
        for c in range(COLS):
            candy = g['grid'][r][c]
            if candy is None:
                continue
            cx, cy = cell_center(r, c)
            scale = candy['s']
            if (r, c) == g['selected']:
                scale *= 1.12 + 0.05 * math.sin(t_ms * 8)
                pygame.draw.rect(screen, (255, 255, 255), (cx - 19, cy - 19, 38, 38), 3,
border_radius=8)
            if hint and (r, c) in hint:
                scale *= 1.0 + 0.1 * math.sin(t_ms * 8)
            draw_candy(candy['t'], int(cx + candy['ox']), int(cy + candy['oy']), scale)
    screen.set_clip(None)

    for p in g['particles']:
        pygame.draw.circle(screen, p['color'], (int(p['x']), int(p['y'])), max(1, int(4 *
p['life'] / 0.5)))
    for fl in g['floaters']:
        text(fl['text'], f_mid, (255, 255, 255), int(fl['x']), int(fl['y']))
    if g['popup']:
        text(g['popup'][0], f_big, (255, 235, 90), W // 2, BY + BOARD_H // 2)

    text('Tap a candy, then a neighbour. Match 3!', f_small, (255, 255, 255), W // 2, BY + BOARD_H
+ 30)
    text(f"Best score: {g.get('best', 0)}", f_small, (255, 255, 255), W // 2, BY + BOARD_H + 56)

    if g['phase'] in ('win', 'lose'):
        shade = pygame.Surface((W, H), pygame.SRCALPHA)
        pygame.draw.rect(shade, (30, 0, 60, 190), (20, 215, 320, 270), border_radius=20)
        screen.blit(shade, (0, 0))
        if g['phase'] == 'win':
            left = g['moves']
            text('LEVEL COMPLETE!', f_big, (255, 235, 90), W // 2, 260)
            draw_stars(W // 2, 330, 3 if left >= 10 else 2 if left >= 5 else 1)
            text(f"Score: {g['score']}", f_mid, (255, 255, 255), W // 2, 390)
            text('Tap for next level', f_mid, (255, 255, 255), W // 2, 440)
        else:
            text('OUT OF MOVES!', f_big, (255, 120, 120), W // 2, 275)
            text(f"Score: {g['score']} / {g['target']}", f_mid, (255, 255, 255), W // 2, 345)
            text('Tap to try again', f_mid, (255, 255, 255), W // 2, 420)


# ============================================================ input
def press(cell, pos):
    g['down'] = (cell, pos)
    sel = g['selected']
    if sel is None:
        g['selected'] = cell
    elif sel == cell:
        g['selected'] = None
    elif abs(sel[0] - cell[0]) + abs(sel[1] - cell[1]) == 1:
        try_swap(sel, cell)
    else:
        g['selected'] = cell
```

```
    g['hint'], g['idle_t'] = None, 0.0


def drag(pos):
    (r, c), start = g['down']
    dx, dy = pos[0] - start[0], pos[1] - start[1]
    if math.hypot(dx, dy) < 18:
        return
    if abs(dx) > abs(dy):
        target = (r, c + (1 if dx > 0 else -1))
    else:
        target = (r + (1 if dy > 0 else -1), c)
    if 0 <= target[0] < ROWS and 0 <= target[1] < COLS:
        try_swap((r, c), target)
    g['down'] = None


async def main():
    setup_level(1)
    running = True
    while running:
        dt = min(clock.tick(60) / 1000, 0.05)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            elif event.type == pygame.MOUSEBUTTONDOWN:
                if g['phase'] in ('win', 'lose'):
                    if g['end_t'] > 0.6:
                        setup_level(g['level'] + 1 if g['phase'] == 'win' else g['level'])
                elif g['phase'] == 'idle':
                    cell = cell_at(event.pos)
                    if cell:
                        press(cell, event.pos)
            elif event.type == pygame.MOUSEMOTION and event.buttons[0]:
                if g['phase'] == 'idle' and g['down']:
                    drag(event.pos)
            elif event.type == pygame.MOUSEBUTTONUP:
                g['down'] = None

        update(dt)
        draw()
        pygame.display.flip()
        await asyncio.sleep(0)       # needed so the game also runs in the browser

    pygame.quit()


asyncio.run(main())
```