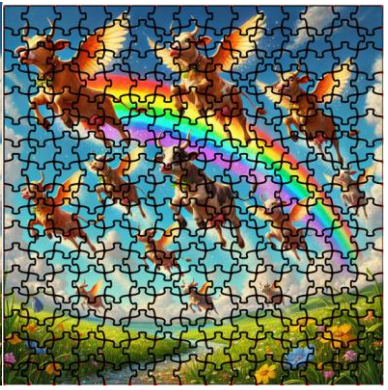
AP 4

```python
import numpy as np
import matplotlib.pyplot as plt
import random
import cv2

image = cv2.imread('./content/fly.webp')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

def bezier_curve(p1, p2, p3, p4, t):
    return (1 - t) ** 3 * p1 + 3 * (1 - t) ** 2 * t * p2 + 3 * (1 - t) * t ** 2 *
p3 + t ** 3 * p4

def generate_grid(width, height, pieces):
    cols = int(np.sqrt(pieces * (width / height)))
    rows = int(np.ceil(pieces / cols))
    cell_w, cell_h = width / cols, height / rows
    points = [(j * cell_w, i * cell_h) for i in range(rows + 1) for j in
range(cols + 1)]
    return points, rows, cols, cell_w, cell_h

def create_knob_curve(start, end, neck=1, width=0.5, height=0.5, pull=0.1):
    x1, y1, x2, y2 = *start, *end
    mx, my = (x1 + x2) / 2, max(y1, y2) + abs(x2 - x1) * height
    ctrls = [
        (x1 + (mx - x1) * neck, y1 + (my - y1) * pull),
        (mx - (mx - x1) * width, my),
        (mx + (x2 - mx) * width, my),
        (x2 - (x2 - mx) * neck, y2 + (my - y2) * pull),
    ]
    t = np.linspace(0, 1, 100)
    curve1 = bezier_curve(x1, ctrls[0][0], ctrls[1][0], mx, t), bezier_curve(y1,
ctrls[0][1], ctrls[1][1], my, t)
    curve2 = bezier_curve(mx, ctrls[2][0], ctrls[3][0], x2, t), bezier_curve(my,
ctrls[2][1], ctrls[3][1], y2, t)
    return np.concatenate([curve1[0], curve2[0]]), np.concatenate([curve1[1],
curve2[1]])

def draw_knobs(ax, points, rows, cols):
    params = dict(neck=1.5, width=0.8, height=0.3, pull=0.2)
    color = "black"
    # horizontal
    for r in range(1, rows):
        for c in range(cols):
            p1, p2 = points[r * (cols + 1) + c], points[r * (cols + 1) + c + 1]
            x, y = create_knob_curve(p1, p2, **params)
            if random.choice([0, 1]): y = [2 * p1[1] - yi for yi in y]
            ax.plot(x, y, color=color, linewidth=1.5)

    # vertical
    for c in range(1, cols):
        for r in range(rows):
            p1, p2 = points[r * (cols + 1) + c], points[(r + 1) * (cols + 1) + c]
            y, x = create_knob_curve((p1[1], p1[0]), (p2[1], p2[0]), **params)
            if random.choice([0, 1]): x = [2 * p1[0] - xi for xi in x]
            ax.plot(x, y, color=color, linewidth=1.5)

def render_puzzle(image, width, height, points, rows, cols):
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.imshow(image)
    ax.set_xlim(0, width)
    ax.set_ylim(height, 0)
    ax.plot([0, width, width, 0, 0], [0, 0, height, height, 0], color="black")
    for x, y in points: ax.plot(x, y, 'ro', markersize=2)
    draw_knobs(ax, points, rows, cols)
    plt.axis('off')
    plt.show()

img_w, img_h = image.shape[1], image.shape[0]
points, rows, cols, cell_w, cell_h = generate_grid(img_w, img_h, 250)
render_puzzle(image, img_w, img_h, points, rows, cols)
```

Input: **Output:**



Code explanation:

Number of pieces of a puzzle:

The number of pieces is defined by the pieces parameter in the generate_grid function. In this case i use 250 pieces. The function divides the image into smaller grid sections based on the number of pieces.

What is the input domain: square, rectangle, circle, ellipse, etc.

The input domain in this code is an image, so a rectangle, the generate_grid function works with a rectangular grid based on the width and height of the image.

What are inputs for curve generation?

The inputs for curve generation are start and end points representing the corners of the puzzle pieces. These points are used to create knobs for the puzzle pieces. parameters like neck, width, height, and pull control the shape and size of the curves.

How can curves be used for puzzle generation?

The curves are used to create the knobs at the edges of the puzzle pieces. These curves are drawn between adjacent grid points, rows and columns, giving each puzzle piece interlocking edges. The curves are defined using the create_knob_curve function, which generates Bezier curves for each edge.

Shape of each piece of a puzzle, how is it generated?

The shape of each piece is primarily a rectangle, but its edges are modified by the curves to make it interlocking. The rectangular grid is divided into smaller cells, and the edges of these cells are then modified by the create_knob_curve function to introduce the curved edges, giving each piece a puzzle shape.

How are data points generated?

They are generated by the generate_grid function. It divides the image's width and height into cells based on the number of puzzle pieces, calculating the number of rows and columns. Then, it creates a list of coordinates for the grid points. These coordinates represent the corners of the puzzle pieces and are used to draw the curves.

Which method can be used for curve generation?

The method used for curve generation is the Bezier curve. The bezier_curve function generates a cubic Bezier curve based on four control points. This curve is used to create the interlocking edges of the puzzle pieces. The create_knob_curve function generates a series of Bezier curves for each edge of the puzzle piece.