Prosjekt 3, The Chaos Game

Problemstilling

- I dette prosjektet skal vi implementere det velkjente “spillet”; The Chaos Game som er en enkel algoritme som produserer fraktaler. Vi begynner enkelt med å se på geometriske figurer, plotting av hjørner og utvider til å plotte det vi kaller for n-gons som er en n-kant på godt norsk :).
- Oppgaven er delt i 4 deler med deloppgaver under hver del
- Part 1: The triangle
- Part 2: Generalizing the Chaos Game
- Part 3: Barnsley Ferns
- Part 4: Variations

Metode og utfordringer

Lager klasse med spillet, forskjellige metoder for å iterere n_gons, starting points, og plotting.

Part 1

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

def triangle(c1, c2):
    ax = c2[0] - c1[0]
    ay = c2[1] - c1[1]

    # calculate last corner from the two other corners.

    alpha = 60.0 / 180 * np.pi

    # Finding new point

    xp = c1[0] + np.cos(alpha) * ax + np.sin(alpha) * ay
    yp = c1[1] + np.sin(-alpha) * ax + np.cos(alpha) * ay
    c3 = np.array([xp, yp])
    c2 = np.array(c2)
    c1 = np.array(c1)
    corners = [c1, c2, c3]
    return corners

# Function to plot and show the triangle
def plot_tri(corner_list, plot_name):
    plt.scatter(*zip(*corner_list))
    plt.title(plot_name)
    plt.grid()
    plt.show()

# Function for weights and corners to be used in the other functions
def coords(corner_list):
    coords = 0
    corners = triangle([1, 0], [0, 0])
    w = np.random.random(3)
    w /= w.sum()

    for i in range(3):
        coords += corners[i] * w[i]
    return coords
```

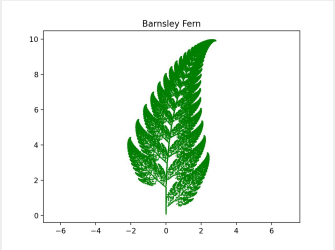
Part 2

```python
class ChaosGame:
    def __init__(self, n, r):
        self.n = n
        self.r = r
        if type(self.n) != int:
            raise TypeError

        if type(self.r) != float:
            raise ValueError

        if self.n < 3:
            raise ValueError

        if self.r < 0 or self.r > 1:
            raise ValueError

        self._generate_ngon()
        self._starting_point()
        self.plot_ngon()

    def _generate_ngon(self):
        self.corners = []
        self.theta = np.linspace(0, 2 * np.pi, self.n + 1)
        self.theta = self.theta[1:]
        for i in self.theta:
            cr = (np.sin(i), np.cos(i))
            self.corners.append(cr)

    def plot_ngon(self):
        plt.scatter(*zip(*self.corners), c="#ff7f0e")
        # plt.scatter(*zip(*self.X_list), c = 'red')

        plt.show()
```

```python
    def _starting_point(self):

        self.X_list = []
        self.X = 0
        w = np.random.random(self.n)
        w /= w.sum()

        for i in range(self.n):
            self.X += np.array(self.corners[i]) * w[i]
        return self.X

    def iterate(self, steps, discard=5):
        if steps<0:
            raise ValueError
        self.steps = steps
        if self.steps < 0:
            raise ValueError
        self.X = np.zeros((self.steps, 2))
        self.newPoints = []
        self.idx = np.zeros(steps)
        for j in range(1, steps):

            random_corner = np.random.randint(self.n)
            self.idx[j] = random_corner
            self.X[j] = self.r * self.X[j - 1] + (1 - self.r) * np.array(
                self.corners[random_corner]
            )
        return self.X, self.idx



    def plot(self, color=False, cmap="jet"):
        if type(color) != bool:
            raise TypeError
        if color == True:
            plt.scatter(*zip(*self.X), s=0.5, c=self.gradient_color, cmap=cmap)
            # plt.scatter(*zip(*self.X), s=0.5, c= self.idx,cmap=cmap)
        else:
            colr = "black"
            plt.scatter(*zip(*self.X), s=0.5, c=colr, cmap=cmap)

    def show(color=False, cmap="jet"):
        plt.axis("equal")
        plt.axis("off")
        plt.show()

    @property
    def gradient_color(self):
        self.C = np.zeros((self.steps))
        for j in range(1, self.steps):
            self.C[j] = (self.C[j - 1] + self.idx[j]) / 2
        return self.C
```

Metode og utfordringer

Part 3, Barnsley Ferns



$$f(x,y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

Call function to do the actual transform based on the formula given in the task

```python
def trans_funcs(self, x, y):

    func_num = [1, 2, 3, 4] # f1, f2, f3 and f4 as shown in the task

    # selection with probability vector for a weighted choice
    function = np.random.choice(func_num, p=[0.01, 0.85, 0.07,
                                             0.07])
    if function == 1:
        return (0, 0.16 * y)
    if function == 2:
        return (0.86 * x + 0.04 * y, -0.04 * x + 0.85 * y + 1.6)
    if function == 3:
        return (0.2 * x - 0.26 * y, 0.23 * x + 0.22 * y + 1.6)
    if function == 4:
        return (-0.15 * x + 0.28 * y, 0.26 * x + 0.24 * y + 0.44)
```

2

```python
def __call__(self, x, y):

    self.x = x
    self.y = y

    # Defining the matrices and vectors and calculating the transform

    self.A = np.array([
    [self.a,self.b],
    [self.c, self.d]
    ])
    self.P = np.array([self.x,self.y])
    self.Q = np.array([self.e,self.f])
    transform = np.matmul(self.A, self.P) + self.Q
    return transform
```

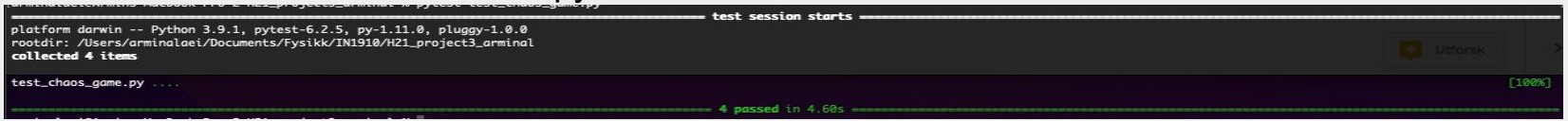
1

```python
    def p_cumulative(self, x, y, steps):
        Points = np.zeros((steps, 2))
        Points[0] = (x, y)
        for i in range(steps):
            Points[i] = self.trans_funcs(Points[i - 1, 0], Points[i - 1, 1]) # x, y coordinates
        return Points

    def plot_fern(self, x, y, steps):
        P = self.p_cumulative(x, y, steps)
        plt.scatter(s=0.2, c='green', *zip(*P))
        plt.axis('equal')
        plt.title('Barnsley Fern')
        plt.show()


if __name__ == '__main__':
    b = AffineTransform()
    b.trans_funcs(x=1, y=1)
    b.p_cumulative(x=1, y=1, steps=50000)
    b.plot_fern(x=1, y=1, steps=50000)
```

3

Fungerer det? Testing

- Har kjørt noen enkle primitive tester for å teste deler av klassen chaosGame
  - testene bestod med pytest:



Utklipp av testene, vi sjekker blant annet for negative verdier som input.

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import numpy as np
from chaos_game import ChaosGame
import pytest


# test 1

def test_chaos_int():
    with pytest.raises(TypeError):
        instance = ChaosGame(-0.5, -2)


# test 2

def test_chaos_valRange():
    with pytest.raises(ValueError):
        instance = ChaosGame(-2, -2)

# test 3
def test_steps():
    with pytest.raises(ValueError):
        instance = ChaosGame(3, 0.5).iterate(-1000)

# test 4
def test_plot_boolInput():
    with pytest.raises(TypeError):
        a= ChaosGame(3, 0.5)
        a.iterate(1000)
        a.plot(color='True')
```