#!/usr/bin/python3

"""

This program renders an animation with the use of the PyPovRay library.

The animation with visualize the mechanism of the second step in the citric acid cycle:

    citrate to isocitrate in the enzym aconitase.

"""

\_\_author\_\_ = "Lisa Hu & Maartje van der Hulst"

\_\_date\_\_ = 2021.12

\_\_version\_\_ = 1.0

# IMPORTS

**import** sys

**from** math **import** pi

**from** pypovray **import** pypovray, pdb, SETTINGS, models

**from** vapory **import** Scene, Texture, Pigment, Finish, Sphere, LightSource, Camera, Background

**import** datetime

# OBJECTS

**class** PovRayObjects:

"""

    Module to make the objects used in PovRay movie

    """

**def** \_\_init\_\_(self):

self.citrate, self.isocitrate = self.get\_molecules()

self.enzyme = self.make\_enzyme()

@staticmethod

**def** make\_enzyme():

"""

        Enzyme object

        """

# Style model for the spheres

sphere\_style = Texture(Pigment('color', [1, 0.7, 0.75], 'filter', 0),

Finish('phong', 0.2, 'reflection', 0.3))

# Spheres

sphere1 = Sphere([30, -1, 0], 5, sphere\_style) # Bottom left

sphere2 = Sphere([35, -1, 0], 5, sphere\_style) # Bottom right

sphere3 = Sphere([30, 4, 0], 5, sphere\_style) # Top left

sphere4 = Sphere([35, 4, 0], 5, sphere\_style) # Top right

**return** [sphere1, sphere2, sphere3, sphere4]

@staticmethod

**def** get\_molecules():

"""

        Chemical components

        """

citrate = pdb.PDBMolecule(f"{SETTINGS.AppLocation}pdb/citrate\_new.pdb",

center=True, offset=[0, 0, 0])

isocitrate = pdb.PDBMolecule(f"{SETTINGS.AppLocation}pdb/isocitrate\_new.pdb",

center=True, offset=[0, 0, 0])

**return** citrate, isocitrate

# FUNCTIONS

**class** PovRayFunctions:

"""

    Module for functions used in PovRay movie

    """

**def** \_\_init\_\_(self, tf\_end):

"""

        Initializing function

        :param tf\_end: List of last frames per scene

        """

self.tf\_end = tf\_end

**def** get\_timesframes(self):

"""

        Create the lists of start frames and duration frames

        """

tf\_start = []

start\_point = 0

**for** i **in** range(len(self.tf\_end)):

# i = [0, 1, 2...]

tf\_start.append(start\_point)

# start\_point = [120, 240, 420...]

start\_point = self.tf\_end[i]

# tf\_start = [0, 120, 240, 420...]

# i = index of the start list

# Subtracting the end time with start time results in duration time

tf\_dur = [self.tf\_end[i] - tf\_start[i] **for** i **in** range(len(tf\_start))]

**return** tf\_start, tf\_dur

@staticmethod

**def** get\_distance(step, duration, distance, start\_time=0):

"""

        Get distances per frame

        :param step: Step in the scene

        :param duration: Duration of the scene in seconds

        :param distance: [x, y, z] vector

        :param start\_time: Start of the movement in the scene, default 0

        """

total\_frames = SETTINGS.RenderFPS \* duration

first\_frame = (step + 1) - SETTINGS.RenderFPS \* start\_time

distances = [x / total\_frames \* first\_frame **for** x **in** distance]

**return** distances

@staticmethod

**def** rotate\_molecule(rotation, molecule, duration, step):

"""

        Function to let the molecule rotate

        :param rotation: Amount of rotation in radials

        :param molecule: Molecule to rotate

        :param duration: Duration of the rotation in frames

        :param step: Step in the rotation

        """

rads = (rotation \* pi / duration) \* step

molecule.rotate([1, 1, 0], rads)

# SCENES

**class** PovRayScenes:

"""

    Scenes used for the povray movie

    """

**def** \_\_init\_\_(self, pr\_objs, pr\_funcs, tf\_start, tf\_dur, tf\_end):

"""

        Initializing function

        :param pr\_objs: PovRayObjects class

        :param pr\_funcs: PovRayFunctions class

        :param tf\_start: List of frames when the scenes start

        :param tf\_dur: List of amounts of frames per scene

        :param tf\_end: List of last frame per scene

        """

self.probj = pr\_objs

self.prfunc = pr\_funcs

self.start = tf\_start

self.dur = tf\_dur

self.end = tf\_end

**def** s1\_citrate\_rotation(self, step):

"""

        First scene: Rotating citrate for visualization

        """

# Create camera and light source

camera = Camera('location', [0, 0, -30], 'look\_at', [0, 0, 0])

lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])

# Get the citrate molecule

citrate = self.probj.citrate

# Let citrate rotate

self.prfunc.rotate\_molecule(2, citrate, self.dur[0] \* 30, step)

# List of objects to render

objects = citrate.povray\_molecule + [lighting]

**return** Scene(camera, objects=objects)

**def** s2\_moving(self, step):

"""

        Second scene: Moving citrate into the enzyme

        """

# Create light source

lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])

# Get the position of the camera

position = self.prfunc.get\_distance(step, self.dur[1], [31, 0, 0])

# Set the z position backwards

position[2] = -30

# Get the position of citrate and the camera look position

looking = self.prfunc.get\_distance(step, self.dur[1], [31, 0, 0])

camera = Camera('location', position, 'look\_at', looking)

# Get citrate and enzyme

enzyme = self.probj.enzyme

citrate = self.probj.citrate

citrate.move\_offset(looking)

# List of objects to render

objects = citrate.povray\_molecule + enzyme + [lighting]

**return** Scene(camera, objects=objects)

**def** s3\_fading\_in(self, step):

"""

        Third scene: Camera moves 'into' the enzyme, light fades

        :return:

        """

# Get the camera location

position = self.prfunc.get\_distance(step, self.dur[2], [0, 0, 25])

position[0] = 32

position[2] -= 30

camera = Camera('location', position, 'look\_at', [30, 0, 0])

# Let the light fade out

intensity = self.prfunc.get\_distance(step, self.dur[2], [-1, -1, -1])

# Outcome of function above is negative -> set positive

intensity[:] = [number + 1 **for** number **in** intensity]

# Light source that fades out

lighting = LightSource([0, 0, -20], 'color', intensity)

# Get enzyme object

enzyme = self.probj.enzyme

# List of objects to render

objects = enzyme + [lighting]

**return** Scene(camera, objects=objects)

**def** s4\_switching(self, step):

"""

        Fourth scene: Located inside the enzyme, visualizing the mechanism of the reaction

        :return:

        """

# Get the objects

background = Background('color', [0.35, 0.16, 0.2])

citrate = self.probj.citrate

camera = Camera('location', [0, 0, -30], 'look\_at', [0, 0, 0])

lighting = LightSource([0, 0, -20], 'color', [1, 1, 1])

# Split respective OH and H atoms from citrate and move away

**if** step <= 90:

offset = self.prfunc.get\_distance(step, self.dur[3] / 3, [3, 0, 0])

oh\_group = citrate.divide([5, 15], 'oh\_group', offset=offset) # Split OH

h\_atom = citrate.divide([13], 'h\_atom', offset=offset) # Split H

# Switch OH and H atoms + rotate OH group

**elif** step <= 180:

oh\_group = citrate.divide([5, 15], 'oh\_group', offset=[6, 0, 0])

h\_atom = citrate.divide([13], 'h\_atom', offset=[6, 0, 0])

# Distances for moving the OH and H up and down

moving\_up = self.prfunc.get\_distance(step, self.dur[3] / 3, [0, 2, -0.55], 3)

moving\_down = self.prfunc.get\_distance(step, self.dur[3] / 3, [0, -1.5, 0.39], 3)

# Move H atom

h\_atom.move\_offset(moving\_down)

# Rotate OH group while moving

oh\_group.move\_offset(moving\_up)

self.prfunc.rotate\_molecule(0.5, oh\_group, (self.dur[3] / 3) \* 30, step - 90)

# Move OH and H atoms back towards molecule

**else**:

oh\_group = citrate.divide([5, 15], 'oh\_group', offset=[6, 2.1, -0.6])

h\_atom = citrate.divide([13], 'h\_atom', offset=[6, -1.5, 0.39])

self.prfunc.rotate\_molecule(0.5, oh\_group, (self.dur[3] / 3) \* 30, 90) # Rotation OH

# Distance for moving OH and H inwards

inwards\_oh = self.prfunc.get\_distance(step, self.dur[3] / 3, [-9.5, 0, 0], 6)

inwards\_h = self.prfunc.get\_distance(step, self.dur[3] / 3, [-4, 0, 0], 6)

# Move OH and H inwards

oh\_group.move\_offset(inwards\_oh)

h\_atom.move\_offset(inwards\_h)

# List of objects to render

objects = citrate.povray\_molecule + oh\_group.povray\_molecule + h\_atom.povray\_molecule + \

[background] + [lighting]

**return** Scene(camera, objects=objects)

**def** s5\_fading\_out(self, step):

"""

        Fifth scene: Isocitrate is made, light fades out, moving out of the enzyme

        """

# Dim lights inside the enzyme

**if** step <= 60:

# Get the objects

isocitrate = self.probj.isocitrate

background = Background('color', [0.35, 0.16, 0.2])

camera = Camera('location', [0, 0, -30], 'look\_at', [0, 0, 0])

# Light fading

intensity = self.prfunc.get\_distance(step, self.dur[4] / 3, [-1, -1, -1])

intensity[:] = [number + 1 **for** number **in** intensity] # Same negative error

# Light fade out

lighting = LightSource([0, 0, -20], 'color', intensity)

# List of objects to render

objects = [background] + [lighting] + isocitrate.povray\_molecule

# Move out of the enzyme

**else**:

# Get the objects

enzyme = self.probj.enzyme

lighting = LightSource([30, 0, -20], 'color', [1, 1, 1])

# Move the camera backwards, out of the enzyme

position = self.prfunc.get\_distance(step, self.dur[4], [0, 0, -30], 2)

position[0] = 35

position[2] -= 5

camera = Camera('location', position, 'look\_at', [30, 0, 0])

# List of objects to render

objects = enzyme + [lighting]

**return** Scene(camera, objects=objects)

**def** s6\_final(self, step):

"""

        Sixth and final scene: Move isocitrate out of the enzyme,

        rotation of isocitrate for visualization

        """

# Get the objects

isocitrate = self.probj.isocitrate

lighting = LightSource([30, 0, -20], 'color', [1, 1, 1])

enzyme = self.probj.enzyme

# Move isocitrate out of the enzyme, camera focuses on isocitrate

**if** step <= 60:

position = self.prfunc.get\_distance(step, self.dur[5] / 3, [20, 0, 0])

position[0] += 31 # Set start position of isocitrate at [31, 0, 0]

isocitrate.move\_offset(position)

camera = Camera('location', [35, 0.0, -25], 'look\_at', position)

# Rotation isocitrate for visualization

**else**:

isocitrate.move\_offset([51, 0, 0])

camera = Camera('location', [35, 0.0, -25], 'look\_at', [51, 0, 0])

self.prfunc.rotate\_molecule(2, isocitrate, ((self.dur[5] / 3) \* 2) \* 30, step - 60)

# List of objects to render

objects = isocitrate.povray\_molecule + enzyme + [lighting]

**return** Scene(camera, objects=objects)

**def** main(step):

"""

    Main function

    """

tf\_end = [120, 240, 420, 690, 870, 1050]

# Scenes 1 2 3 4 5 6

# Initialize functions

probj = PovRayObjects()

prfunc = PovRayFunctions(tf\_end)

# Get the time frames

tf\_start, tf\_dur = prfunc.get\_timesframes()

tf\_dur[:] = [number / 30 **for** number **in** tf\_dur]

# Initialize the scenes

prscenes = PovRayScenes(probj, prfunc, tf\_start, tf\_dur, tf\_end)

# Create the scenes according to step

**if** step <= tf\_end[0]:

scene = prscenes.s1\_citrate\_rotation(step)

**elif** step <= tf\_end[1]:

scene = prscenes.s2\_moving(step - tf\_end[0])

**elif** step <= tf\_end[2]:

scene = prscenes.s3\_fading\_in(step - tf\_end[1])

**elif** step <= tf\_end[3]:

scene = prscenes.s4\_switching(step - tf\_end[2])

**elif** step <= tf\_end[4]:

scene = prscenes.s5\_fading\_out(step - tf\_end[3])

**else**:

scene = prscenes.s6\_final(step - tf\_end[4])

**return** scene

**if** \_\_name\_\_ == "\_\_main\_\_":

**for** i **in** range(1000, 1050):

pypovray.render\_scene\_to\_png(main, i)

pypovray.render\_scene\_to\_png(main, int(sys.argv[1]))