Project 2: A Solar System, A Clock, A Windmill

20171166 정성환

Github url: https://github.com/It-could-happen/Introduction-to-Visual-Media-Programming.git

Draw a small solar system in 2D composed of Sun, Venus, Earth with the Moon, Saturn with the ring and the moon Titan.

- try to use real periods of rotation

- design an elliptical orbit for Saturn (exaggerate; we are not simulating the real solar system)

- Use some real photos for reality enhancement.

- add a starship Enterprise wandering around.

- use a large size screen.

- put some background stars, twinkling.

2. Code

```
import pygame
import math
import numpy as np
import os

# 게임 윈도우 크기
WINDOW_WIDTH = 1200
WINDOW_HEIGHT = 900
CENTER_X = WINDOW_WIDTH/2
CENTER_Y = WINDOW_HEIGHT/2

#행성들의 물리량 (주기, 장반경, 궤도이심률) 지구의 값이 1일때 기준이기 때문에 지구의 값을 곱해주었다
EARTH_T = 1.
EARTH_A = 200.
EARTH_E = 0.0167086

MERCURY_T = 0.24 * EARTH_T
MERCURY_A = 0.39 * EARTH_A
```

```
MERCURY_E = 0.2056409

VENUS_T = 0.61 * EARTH_T
VENUS_A = 0.7 * EARTH_A
VENUS_E = 0.006764

MARS_T = 1.88 * EARTH_T
MARS_A = 1.52 * EARTH_A
MARS_E = 0.0934

JUPITER_T = 11.862 * EARTH_T
JUPITER_A = 5. * EARTH_A
JUPITER_E = 0.0489044

SATURN_T = 29.458 * EARTH_T
SATURN_A = 2.2 * EARTH_A  #원래 장반경이 10배 차이이지만 화면 내 표시를 위해 축소
SATURN_E = 0.0532927
```

It defined the basic window size, center point, and the unique physical quantities of each planet. By defining other planets in this way, it becomes easy to add other planets.

```
class planet:
    def __init__(self, T, A, E, scr, image, r = 10, ):
        self.T = T
        self.A = A
        self.E = E
        self.B = (1-self.E)/(1+self.E) * self.A #이심률 공식을 활용한 단반경 계산

        self.image = image
        self.Radius = r
        self.location = np.array([0,0,1])
        self.screen = scr

    def update(self, deg):
        degree = deg / self.T
        x = int(math.cos(degree * 2 * math.pi/360) * self.A)+CENTER_X
        y = int(math.sin(degree * 2 * math.pi/360) * self.B)+CENTER_Y
```

```
        self.location = np.array([x,y,1])

    def draw(self):
        bx = self.image.get_width() / 2
        by = self.image.get_height() / 2
        self.screen.blit(self.image, (self.location[0]-bx, self.location[1]-by))
```

Planet class is defined moving object. Moving planets are intrinsic values. The reason for receiving the radius (period, long radius, orbital eccentricity, planet image, radius) is that I was just trying to draw it as a circle in case it would be cumbersome to put the image, but it was easier to put the image than I thought, so I just put it in.

Planet.update() is change the location.

The position of each planet is determined by the long radius, short radius, orbital eccentricity according to the definition of an ellipse, and this is stored as a numpy array.

The Playnet.draw() function outputs the corresponding planet object to the screen that receives the input. Since it uses an image, bx, by calculations are required to correct the center position.

```
# 항성 class 배경의 별 들을 생성
class star:
    def __init__(self, scr, x , y ,r, image = None):
        self.x = x
        self.y = y
        self.twincle_timer = np.random.randint(20)
        self.radius = r
        self.screen = scr
        self.color = WHITE
        self.image = image
        self.center = np.array([self.x, self.y, 1])

    def update(self,):
        self.twincle_timer += 1
        if self.twincle_timer > 20:
```

```
            self.twincle_timer = 0

    def drawStar(self, sunflag = 0):

        if sunflag == 0:
            if self.twincle_timer<5:
                self.color = BLACK
            else:
                self.color =WHITE
                pygame.draw.circle(self.screen, self.color, [self.x, self.y], self.radius)
        else:
            bx = self.image.get_width() / 2
            by = self.image.get_height() / 2
            self.screen.blit(self.image, (self.x - bx, self.y - by))
```

The star class defines a fixed object. Since these do not need to move, the Sun and the star have produced them with the same Cl Ass, which is also consistent with scientific fact. The twinkle of the stars is embodied through twincle_timer. If we increment the value of the timer by 1 in the Update() function, when the timer value is in the range of 0 to less than 5 at draw, the star is printed black and is not visible on the screen, which appears to be twinkling to our eyes. That's only one-fifth of the time it actually goes off.

```
#우주선 정의
class ship:
    def __init__(self,x = np.random.randint(WINDOW_WIDTH), y = np.random.randint(WINDOW_HEIGHT), src = None, image = None):
        self.x = x
        self.y = y
        self.dx = 0
        self.dy = 0
        self.screen = src
        self.image = image

    def update(self,):
        self.x += self.dx
```

```
            self.y += self.dy

        def drawShip(self,):
            self.screen.blit(self.image, (self.x, self.y))
```

The spacecraft simply used the keyboard to define the objects to move.

```
#이동 행렬 생성함수
def Tmat(a,b):
    H = np.eye(3)
    H[0,2] = a
    H[1,2] = b
    return H

#회전 행렬 생성 함수
def Rmat(deg):
    radian = np.deg2rad(deg)
    c = np.cos(radian)
    s = np.sin(radian)
    R = np.array([[c,-s, 0], [s, c, 0], [0, 0, 1]])
    return R

#3차원 요소 제거 함수
def remove3thDim(pp):
    q = pp[0:2, :].T
    return q
```

The matrix generation function for positional transformation is the same as the previous robotic arm project.

```
def main():
    # Pygame 초기화
    pygame.init()

    # 윈도우 제목
    pygame.display.set_caption("robot arm by 20171166")
```

```
# 윈도우 생성
screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))

# 게임 화면 업데이트 속도
clock = pygame.time.Clock()

#이미지 경로 설정
current_path = os.path.dirname(__file__)
assets_path = os.path.join(current_path, 'assets')

# 게임 종료 전까지 반복
done = False      # 게임이 진행중인지 확인하는 변수
font = pygame.font.SysFont('FixedSys', 24, True, False)

# done이 True라면 게임이 계속 진행중이라는 의미
degree = 0
```

It's a basic pygame skeleton.

```
# 우주선 생성
shipImage = pygame.image.load(os.path.join(assets_path, "spaceship.png"))
shipImage = pygame.transform.scale(shipImage, (50, 70))
myShip = ship(400,400, screen, shipImage )

# 태양 생성
sunImage = pygame.image.load(os.path.join(assets_path, "sun.png"))
sunImage = pygame.transform.scale(sunImage, (130, 100))
sun = star(screen, CENTER_X, CENTER_Y, 50, sunImage)

# 행성 생성
earthImage = pygame.image.load(os.path.join(assets_path, "earth.png"))
earthImage = pygame.transform.scale(earthImage, (70, 70))
earth = planet(EARTH_T, EARTH_A, EARTH_E, screen, earthImage, 100)

venusImage = pygame.image.load(os.path.join(assets_path, "venus.png"))
```

```
    venusImage = pygame.transform.scale(venusImage, (120, 60))
    venus = planet(VENUS_T, VENUS_A, VENUS_E, screen, venusImage, 100)

    saturnImage = pygame.image.load(os.path.join(assets_path, "saturn.png"))
    saturnImage = pygame.transform.scale(saturnImage, (100, 100))
    saturn = planet(SATURN_T, SATURN_A, SATURN_E, screen, saturnImage, 100)

    moonImage = pygame.image.load(os.path.join(assets_path, "moon.png"))
    moonImage = pygame.transform.scale(moonImage, (30, 20))
    mx = moonImage.get_width()/2
    my = moonImage.get_height()/2

    titanImage = pygame.image.load(os.path.join(assets_path, "titan.png"))
    titanImage = pygame.transform.scale(titanImage, (30, 30))
    tx = titanImage.get_width()/2
    ty = titanImage.get_height()/2

    text1 = font.render("Press the direction key to move rocket", True, WHITE)


    # 배경의 별 생성
    starlist = []
    for i in range(100):
        newstar = star(screen,np.random.randint(WINDOW_WIDTH) ,
np.random.randint(WINDOW_HEIGHT), np.random.randint(1,4))
        starlist.append(newstar)
```

This is the part that creates each instance and gets the image path.

```
# 게임 반복 구간
    while not done: # 게임이 진행되는 동안 계속 반복 작업을 하는 while 루프
        # 이벤트 반복 구간
        for event in pygame.event.get():
            # 어떤 이벤트가 발생했는지 확인
            if event.type == pygame.QUIT:
                done = True # 반복을 중단시켜 게임 종료
            elif event.type == pygame.KEYDOWN:
```

```
            if event.key == pygame.K_F1:
                done = True
        if event.type == pygame.KEYUP:
            myShip.dx = 0
            myShip.dy = 0

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        myShip.dx = -2
    if keys[pygame.K_RIGHT]:
        myShip.dx = 2
    if keys[pygame.K_UP]:
        myShip.dy = -2
    if keys[pygame.K_DOWN]:
        myShip.dy = 2

    # 게임 로직 구간
    degree+=1

    # 화면 삭제 구간

    # 윈도우 화면 채우기
    screen.fill(BLACK)

    # 화면 그리기 구간
    for i in range(100):
        starlist[i].drawStar(0)
        starlist[i].update()

    screen.blit(text1, (50,50))


    myShip.drawShip()
    sun.drawStar(1)
    earth.draw()
    venus.draw()
    saturn.draw()
```

```
            #위성 궤도 연산
                #위성 정의
            moon = np.array([0,0,1])
            titan = np.array([0,0,1])

            MoonH = Tmat(-mx,   -my) @ Tmat(earth.location[0], earth.location[1]) @ Rmat(degree/
(27/365)) @ Tmat(50, 0) @Tmat( -mx, -my)
            print(MoonH.shape)
            print(moon.T)
            moon = (MoonH @ moon.T)[0:2 :].T

            TitanH = Tmat( -tx,-ty ) @ Tmat(saturn.location[0], saturn.location[1]) @ Rmat(degree /
(16 / 365)) @ Tmat(50, 0)
            titan = (TitanH @ titan.T)[0:2 :].T

            screen.blit(moonImage, moon)
            screen.blit(titanImage, titan)

            # 화면 업데이트
            pygame.display.flip()
            myShip.update()
            earth.update(degree)
            venus.update(degree)
            saturn.update(degree)



            # 초당 60 프레임으로 업데이트
            clock.tick(20)

        # 게임 종료


if __name__ == "__main__":
        main()
        pygame.quit()
```

Repeating intervals are not difficult. Since the required functions were defined internally in the above CLASS, only the acceptance of input through the keyboard and the adjustment of the position of

the satellite were required within the main function.

The position of the satellite was implemented by defining a transformation matrix as the product of the movement matrix and the rotation matrix and adding this to the position of each planet. This will require a better approach, as it is cumbersome to reset the position of the satellite every frame.

To reflect the orbital period of each planet and moon, we divided the angle of application in each rotation matrix by period, reflecting one revolution when the Earth rotates 360 degrees. With this, each planet reflects a relatively accurate real-world orbital period.