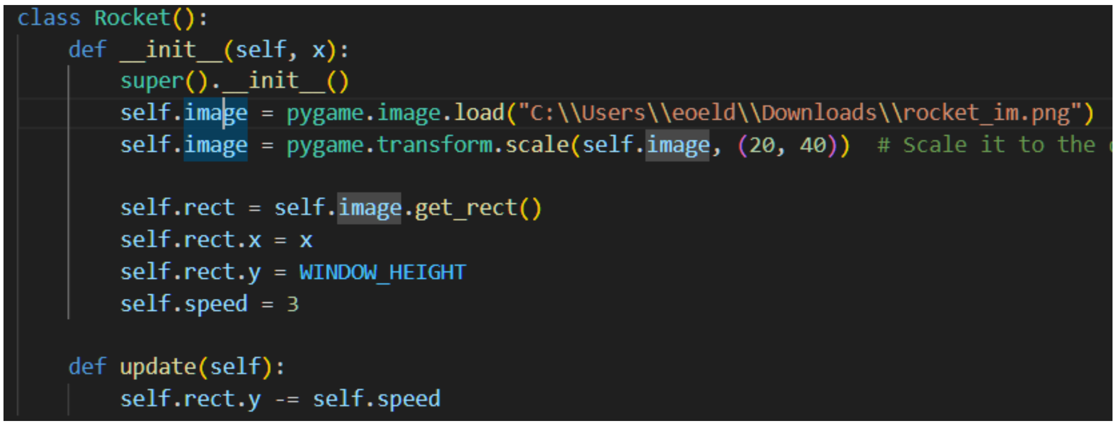
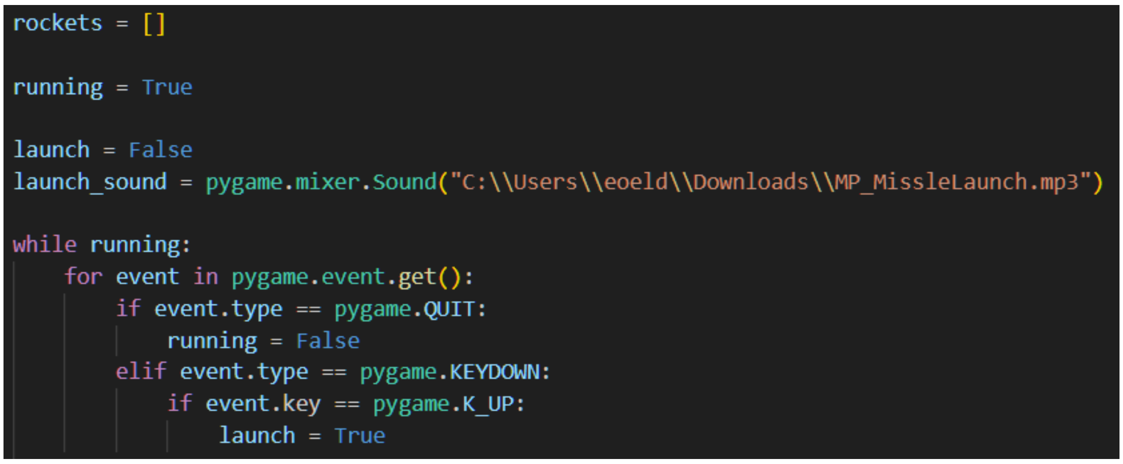
Project 1



```
class Rocket():
    def __init__(self, x):
        super().__init__()
        self.image = pygame.image.load("C:\\Users\\eoeld\\Downloads\\rocket_im.png")
        self.image = pygame.transform.scale(self.image, (20, 40))  # Scale it to the 

        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = WINDOW_HEIGHT
        self.speed = 3

    def update(self):
        self.rect.y -= self.speed
```

The Rocket class is defined to represent the rockets that will be launched. The class inherits from pygame.sprite.Sprite, which is a simple base class for visible game objects. The Rocket class has an __init__ method that loads an image, scales it, and sets its initial position and speed. It also has an update method that moves the rocket upwards by decreasing its y-coordinate.



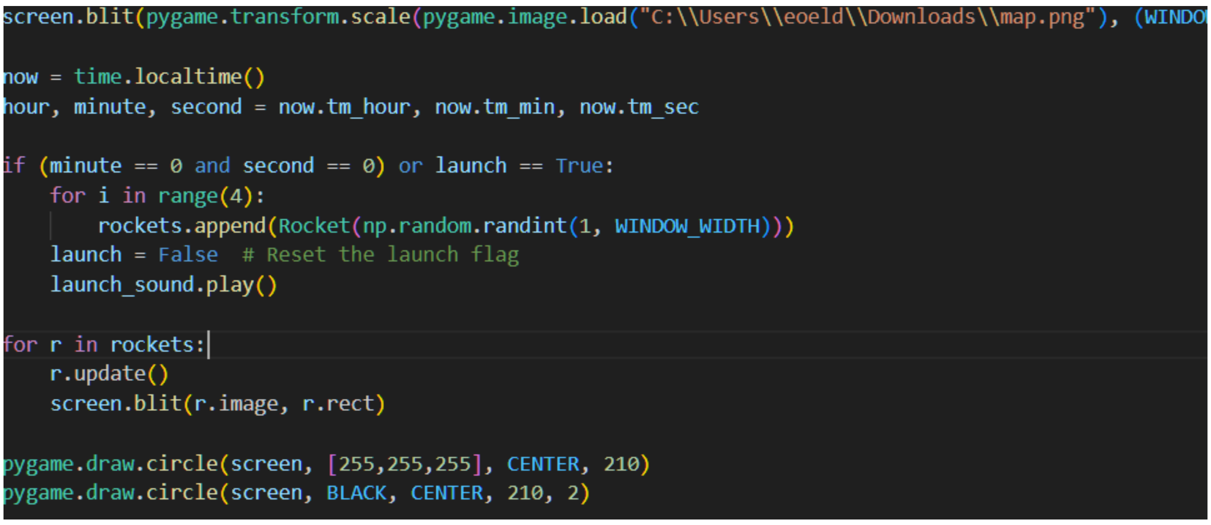
```
rockets = []

running = True

launch = False
launch_sound = pygame.mixer.Sound("C:\\Users\\eoeld\\Downloads\\MP_MissleLaunch.mp3")

while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                launch = True
```

An empty list rockets is created to hold the rocket objects.

launch variable is created to check the rocket is launched.

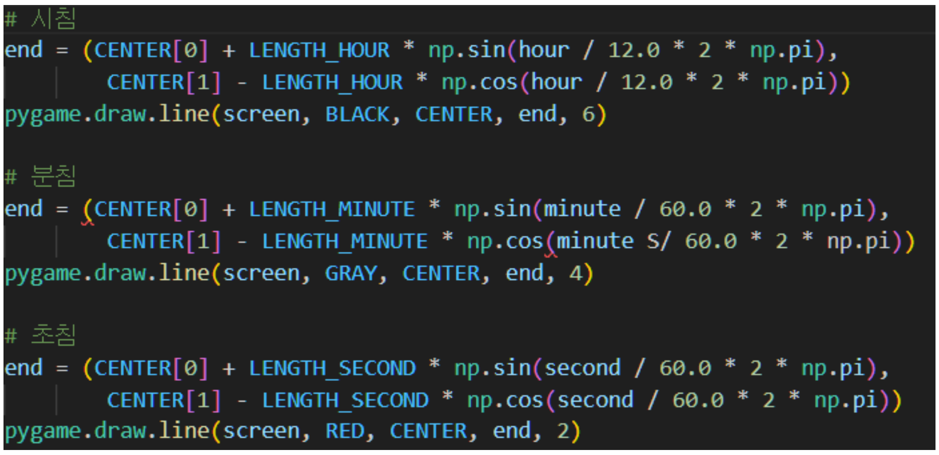
Import sound of rocket launch and save it in launch_sound variable.

The main game loop starts with ‘while running’. Inside the loop, the code first checks for events (like the user closing the window or pressing a key) and reacts accordingly. If the user presses the UP key, the launch flag is set to True.



```python
screen.blit(pygame.transform.scale(pygame.image.load("C:\\Users\\eoeld\\Downloads\\map.png"), (WINDO

now = time.localtime()
hour, minute, second = now.tm_hour, now.tm_min, now.tm_sec

if (minute == 0 and second == 0) or launch == True:
    for i in range(4):
        rockets.append(Rocket(np.random.randint(1, WINDOW_WIDTH)))
    launch = False  # Reset the launch flag
    launch_sound.play()

for r in rockets:
    r.update()
    screen.blit(r.image, r.rect)

pygame.draw.circle(screen, [255,255,255], CENTER, 210)
pygame.draw.circle(screen, BLACK, CENTER, 210, 2)
```

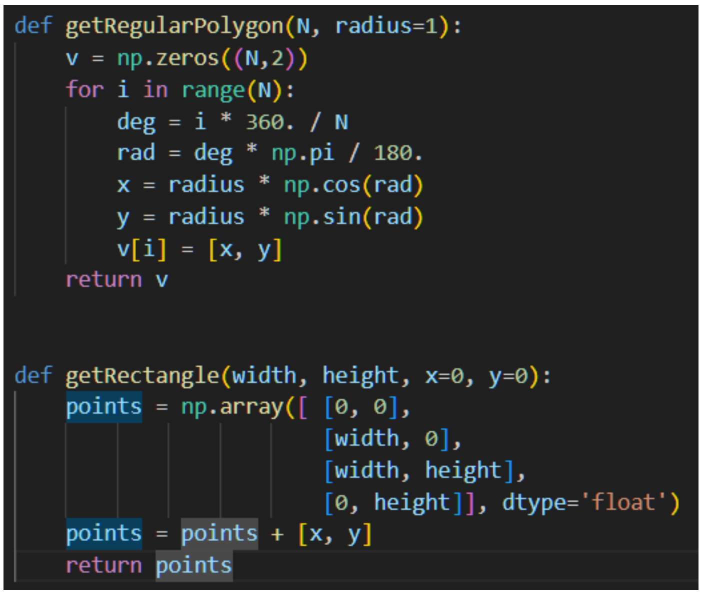
If the current minute and second are both zero (i.e., it's the start of an hour) or if the launch flag is True, new rockets are created and added to the rockets list, and a launch sound is played.
For each rocket in the rockets list, the rocket's update method is called to move the rocket, and then the rocket is drawn on the screen.



```python
# 시침
end = (CENTER[0] + LENGTH_HOUR * np.sin(hour / 12.0 * 2 * np.pi),
        CENTER[1] - LENGTH_HOUR * np.cos(hour / 12.0 * 2 * np.pi))
pygame.draw.line(screen, BLACK, CENTER, end, 6)

# 분침
end = (CENTER[0] + LENGTH_MINUTE * np.sin(minute / 60.0 * 2 * np.pi),
        CENTER[1] - LENGTH_MINUTE * np.cos(minute S/ 60.0 * 2 * np.pi))
pygame.draw.line(screen, GRAY, CENTER, end, 4)

# 초침
end = (CENTER[0] + LENGTH_SECOND * np.sin(second / 60.0 * 2 * np.pi),
        CENTER[1] - LENGTH_SECOND * np.cos(second / 60.0 * 2 * np.pi))
pygame.draw.line(screen, RED, CENTER, end, 2)
```

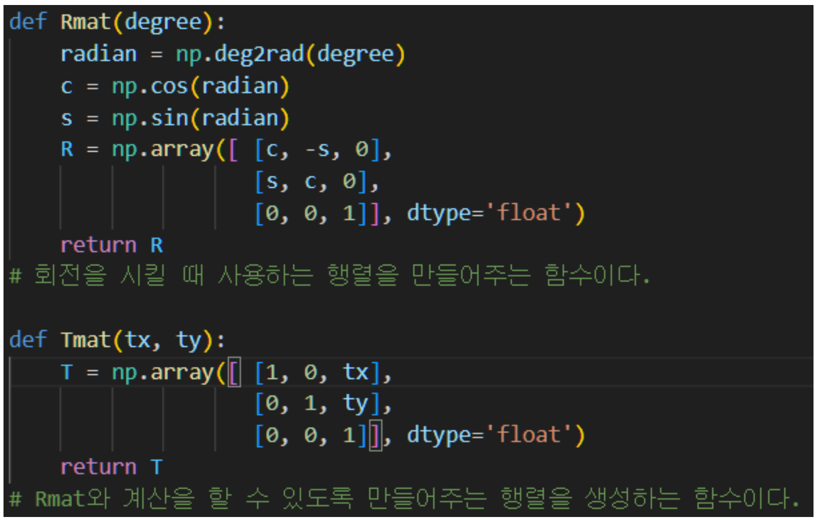
The code draws a clock in the center of the screen. The clock consists of a circle and three hands (for hours, minutes, and seconds). The positions of the hands are calculated based on the current time.

Project 2



```python
def getRegularPolygon(N, radius=1):
    v = np.zeros((N,2))
    for i in range(N):
        deg = i * 360. / N
        rad = deg * np.pi / 180.
        x = radius * np.cos(rad)
        y = radius * np.sin(rad)
        v[i] = [x, y]
    return v


def getRectangle(width, height, x=0, y=0):
    points = np.array([ [0, 0],
                        [width, 0],
                        [width, height],
                        [0, height]], dtype='float')
    points = points + [x, y]
    return points
```

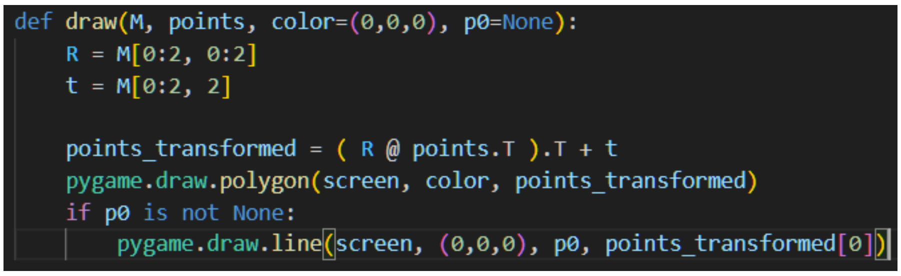
These two functions are used to make Regular polygon, and rectangular.



```python
def Rmat(degree):
    radian = np.deg2rad(degree)
    c = np.cos(radian)
    s = np.sin(radian)
    R = np.array([ [c, -s, 0],
                   [s, c, 0],
                   [0, 0, 1]], dtype='float')
    return R
# 회전을 시킬 때 사용하는 행렬을 만들어주는 함수이다.

def Tmat(tx, ty):
    T = np.array([ [1, 0, tx],
                   [0, 1, ty],
                   [0, 0, 1]], dtype='float')
    return T
# Rmat와 계산을 할 수 있도록 만들어주는 행렬을 생성하는 함수이다.
```

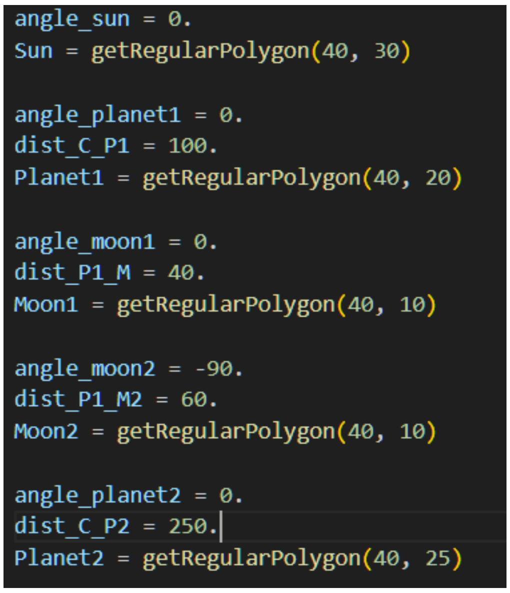
Rmat function creates a rotation matrix. The rotation matrix is a matrix that is used to perform a rotation. The function first converts this angle to radians since the trigonometric functions in the numpy expect angles in radians. It then calculates the cosine and sine of the angle. This matrix can be used to rotate a point or a vector in 2D space around the origin. The third row and column are added to make the matrix compatible with the translation matrix.

Tmat function can be used to translate a point or a vector in 2D space. The third row and column are added to make the matrix compatible with the rotation matrix.



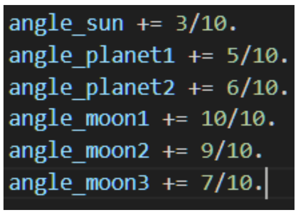
```python
def draw(M, points, color=(0,0,0), p0=None):
    R = M[0:2, 0:2]
    t = M[0:2, 2]

    points_transformed = ( R @ points.T ).T + t
    pygame.draw.polygon(screen, color, points_transformed)
    if p0 is not None:
        pygame.draw.line(screen, (0,0,0), p0, points_transformed[0])
```

By using the matrix created from above functions, draw function draw polygon in screen.



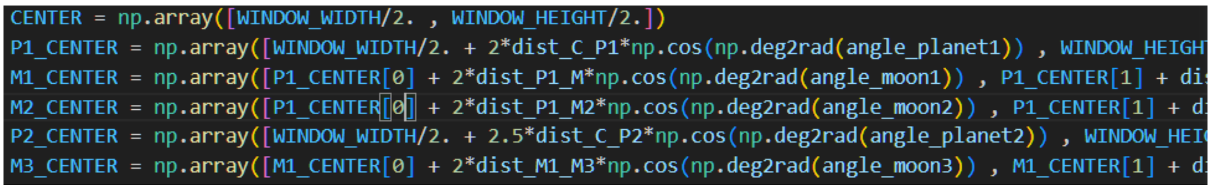
```python
angle_sun = 0.
Sun = getRegularPolygon(40, 30)

angle_planet1 = 0.
dist_C_P1 = 100.
Planet1 = getRegularPolygon(40, 20)

angle_moon1 = 0.
dist_P1_M = 40.
Moon1 = getRegularPolygon(40, 10)

angle_moon2 = -90.
dist_P1_M2 = 60.
Moon2 = getRegularPolygon(40, 10)

angle_planet2 = 0.
dist_C_P2 = 250.
Planet2 = getRegularPolygon(40, 25)
```

Define the variables to make planet, and moon. Angle_() is the angle of the planet or moon. Dist_()_() is distance between two planet or moon. Make the planet polygon by using getRegularPolygon function.



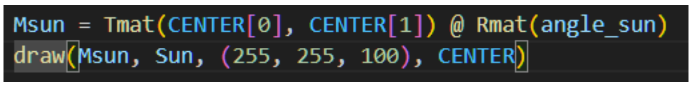
```python
angle_sun += 3/10.
angle_planet1 += 5/10.
angle_planet2 += 6/10.
angle_moon1 += 10/10.
angle_moon2 += 9/10.
angle_moon3 += 7/10.
```

These lines are updating the angles of the sun, planets, and moons. The angles are used to calculate the positions of these objects in the following lines. The numbers being added to the angles determine the speed of rotation for each object. For example, the sun rotates slower than the planets and moons because a smaller number is being added to its angle.



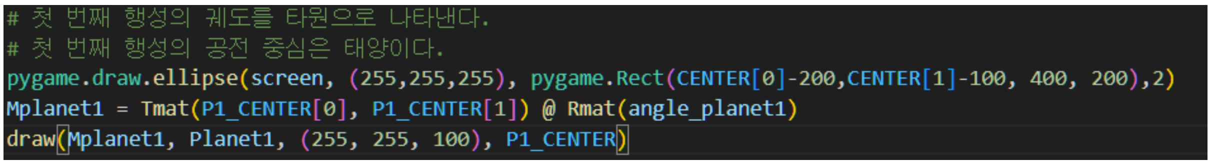
```python
CENTER = np.array([WINDOW_WIDTH/2. , WINDOW_HEIGHT/2.])
P1_CENTER = np.array([WINDOW_WIDTH/2. + 2*dist_C_P1*np.cos(np.deg2rad(angle_planet1)) , WINDOW_HEIGHT
M1_CENTER = np.array([P1_CENTER[0] + 2*dist_P1_M*np.cos(np.deg2rad(angle_moon1)) , P1_CENTER[1] + di
M2_CENTER = np.array([P1_CENTER[0] + 2*dist_P1_M2*np.cos(np.deg2rad(angle_moon2)) , P1_CENTER[1] + d
P2_CENTER = np.array([WINDOW_WIDTH/2. + 2.5*dist_C_P2*np.cos(np.deg2rad(angle_planet2)) , WINDOW_HEI
M3_CENTER = np.array([M1_CENTER[0] + 2*dist_M1_M3*np.cos(np.deg2rad(angle_moon3)) , M1_CENTER[1] + d
```

These lines are setting the position of the sun to be in the center of the screen. The M1_CENTER, M2_CENTER, P2_CENTER, and M3_CENTER lines are doing similar calculations to find the positions of the moons and the second planet.



```python
Msun = Tmat(CENTER[0], CENTER[1]) @ Rmat(angle_sun)
draw(Msun, Sun, (255, 255, 100), CENTER)
```

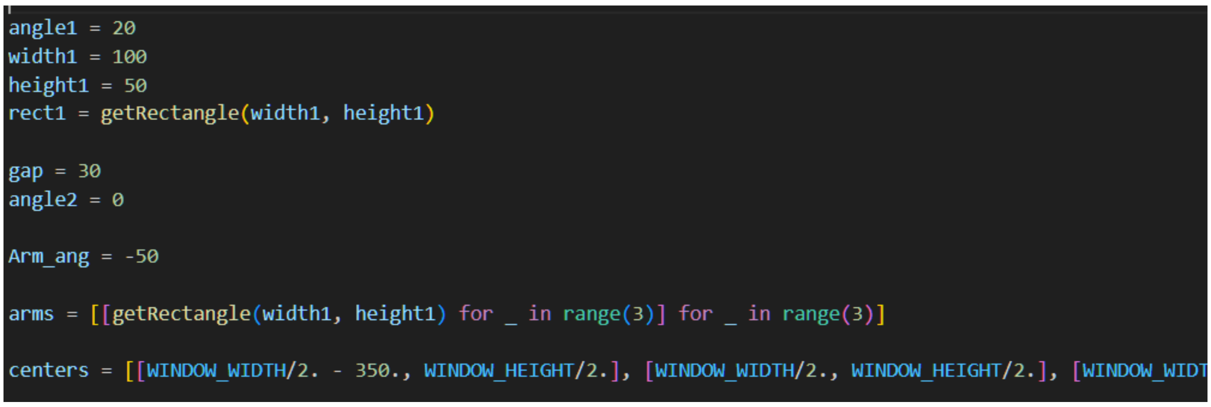
This line is creating a transformation matrix for the sun. The transformation matrix is used to rotate and translate the sun's position. The @ operator is used to perform matrix multiplication. And draw function is drawing the sun on the screen. The draw function takes a transformation matrix, a set of points, a color, and a center point as arguments.



```python
# 첫 번째 행성의 궤도를 타원으로 나타낸다.
# 첫 번째 행성의 공전 중심은 태양이다.
pygame.draw.ellipse(screen, (255,255,255), pygame.Rect(CENTER[0]-200,CENTER[1]-100, 400, 200),2)
Mplanet1 = Tmat(P1_CENTER[0], P1_CENTER[1]) @ Rmat(angle_planet1)
draw(Mplanet1, Planet1, (255, 255, 100), P1_CENTER)
```

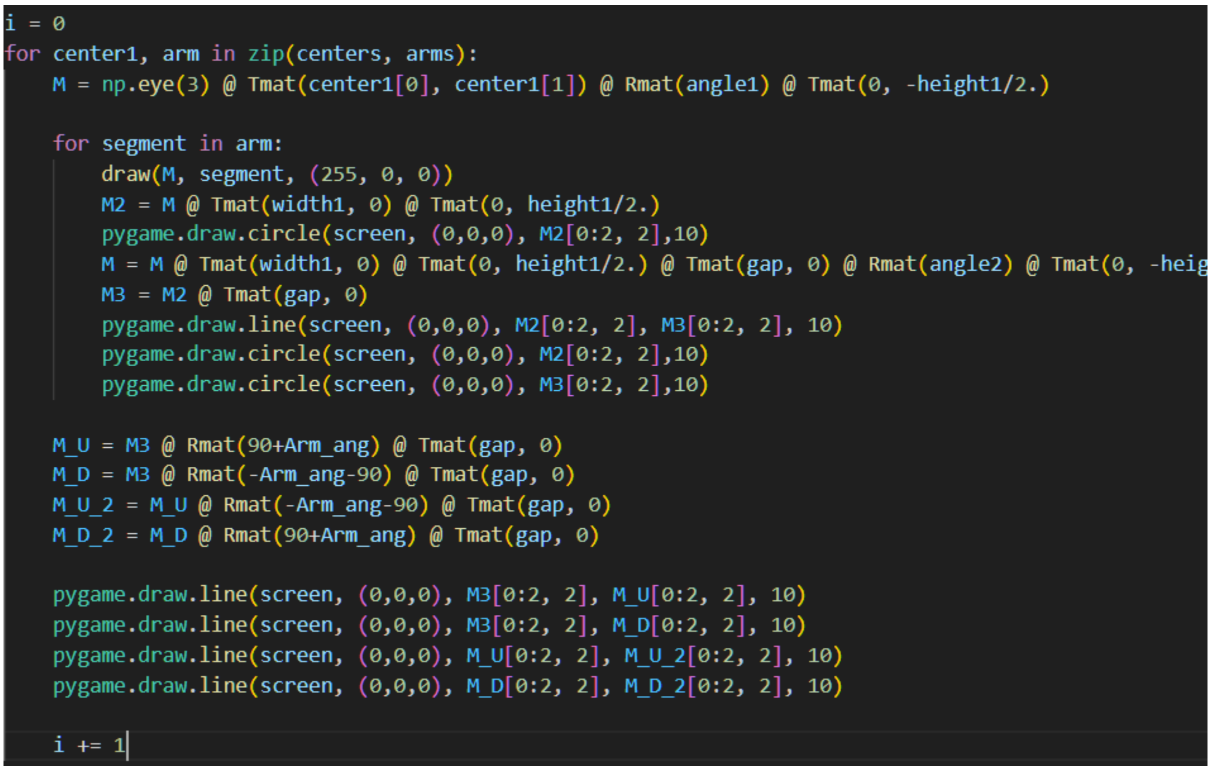
The pygame.draw.ellipse lines are drawing the orbits of the planets and moons. The pygame.Rect function is used to specify the position and size of the ellipse. The remaining lines are creating transformation matrices and drawing the planets and moons in a similar way to the sun.

Project 3



```python
angle1 = 20
width1 = 100
height1 = 50
rect1 = getRectangle(width1, height1)

gap = 30
angle2 = 0

Arm_ang = -50

arms = [[getRectangle(width1, height1) for _ in range(3)] for _ in range(3)]

centers = [[WINDOW_WIDTH/2. - 350., WINDOW_HEIGHT/2.], [WINDOW_WIDTH/2., WINDOW_HEIGHT/2.], [WINDOW_WIDT
```

angle1 sets the initial angle of the first joint of the robotic arms to 20 degrees.
Width1 and height1 set the dimensions of the segments of the robotic arms. Each arm is made up of rectangular segments.
By using getRectangle, rect1 has a rectangle that represents a segment of the robotic arm using the dimensions specified above.
gap sets the distance between the segments of the robotic arms.
angle2 sets the initial angle of the second joint of the robotic arms to 0 degrees.
Arm_ang sets the initial angle of the robotic arms to -50 degrees.
List ‘arms’ creates a 2D list that represents the three robotic arms. Each arm is represented by a list of three rectangles.
centers sets the initial positions of the three robotic arms on the screen. The arms are positioned horizontally across the middle of the screen, with a distance of 350 units between them.



```
i = 0
for center1, arm in zip(centers, arms):
    M = np.eye(3) @ Tmat(center1[0], center1[1]) @ Rmat(angle1) @ Tmat(0, -height1/2.)

    for segment in arm:
        draw(M, segment, (255, 0, 0))
        M2 = M @ Tmat(width1, 0) @ Tmat(0, height1/2.)
        pygame.draw.circle(screen, (0,0,0), M2[0:2, 2],10)
        M = M @ Tmat(width1, 0) @ Tmat(0, height1/2.) @ Tmat(gap, 0) @ Rmat(angle2) @ Tmat(0, -heig
        M3 = M2 @ Tmat(gap, 0)
        pygame.draw.line(screen, (0,0,0), M2[0:2, 2], M3[0:2, 2], 10)
        pygame.draw.circle(screen, (0,0,0), M2[0:2, 2],10)
        pygame.draw.circle(screen, (0,0,0), M3[0:2, 2],10)

    M_U = M3 @ Rmat(90+Arm_ang) @ Tmat(gap, 0)
    M_D = M3 @ Rmat(-Arm_ang-90) @ Tmat(gap, 0)
    M_U_2 = M_U @ Rmat(-Arm_ang-90) @ Tmat(gap, 0)
    M_D_2 = M_D @ Rmat(90+Arm_ang) @ Tmat(gap, 0)

    pygame.draw.line(screen, (0,0,0), M3[0:2, 2], M_U[0:2, 2], 10)
    pygame.draw.line(screen, (0,0,0), M3[0:2, 2], M_D[0:2, 2], 10)
    pygame.draw.line(screen, (0,0,0), M_U[0:2, 2], M_U_2[0:2, 2], 10)
    pygame.draw.line(screen, (0,0,0), M_D[0:2, 2], M_D_2[0:2, 2], 10)

    i += 1
```

The loop above iterates over each robotic arm and its corresponding center position.

M is the result of calculation of the transformation matrix M for the first joint of the current robotic arm. The transformation matrix is a combination of a translation to the center position of the arm (Tmat(center1[0], center1[1])), a rotation by angle1 degrees (Rmat(angle1)), and a translation by half the height of the arm segment in the negative y-direction (Tmat(0, -height1/2.)).

The inner loop for segment in arm iterates over each segment of the current robotic arm. For each segment, it draws the segment (draw(M, segment, (255, 0, 0))), calculates the transformation matrix for the next joint (M2 = M @ Tmat(width1, 0) @ Tmat(0, height1/2.)), and draws a circle at the joint position (pygame.draw.circle(screen, (0,0,0), M2[0:2, 2],10)). Then, it updates the transformation matrix M for the next segment of the arm.

After drawing all segments of the arm, it calculates the transformation matrices for the upper (M_U) and lower (M_D) parts of the arm, and for the second joints of the upper (M_U_2) and lower (M_D_2) parts. These transformation matrices are used to draw the upper and lower parts of the arm and their second joints.

Finally, it draws lines representing the upper and lower parts of the arm and their second joints, and increments the counter i.