**Документација за кодот:**

За оваа проектна се користи pybullet physics engine кој овозможува реална симулација на физика и движење на роботи.

import pybullet

Најпрво се поврзуваме со серверот кој ги овозможува симулациите:

physicsClient = pybullet.connect(pybullet.GUI)  
pybullet.setAdditionalSearchPath(pybullet\_data.getDataPath())

Потоа се мести камерата за симулацијата:

pybullet.resetDebugVisualizerCamera(cameraDistance = 2,  
 cameraYaw = 10,  
 cameraPitch = 5,  
 cameraTargetPosition = [0, -4, 1])

Креираме колизиски објекти – сфери со соодветни радиуси:

sphere = pybullet.createCollisionShape(pybullet.GEOM\_SPHERE, radius = 0.6)  
sphere2 = pybullet.createCollisionShape(pybullet.GEOM\_SPHERE, radius = 1)  
sphere3 = pybullet.createCollisionShape(pybullet.GEOM\_SPHERE, radius = 0.2)

* 0.6 за Sun, 1 за Neutron и 0.2 за планетата соодветно со нивните позиции на координатната оска.
* spherePos1 = [2,-2,0.2]  
  spherePos2 = [-2.5,-1,2.5]  
  spherePos3 = [0,0,0]

Колизиските објекти ги креираме сега со VisualShape:

visualShapeId = pybullet.createVisualShape(shapeType = pybullet.GEOM\_SPHERE,  
 rgbaColor = [1,0.47,0.97,1],  
 radius = 0.2  
 )  
visualShapeId2 = pybullet.createVisualShape(shapeType = pybullet.GEOM\_SPHERE,  
 rgbaColor = [0.26,0.8,0.9,1],  
 radius = 1  
 )  
visualShapeId3 = pybullet.createVisualShape(shapeType = pybullet.GEOM\_SPHERE,  
 rgbaColor = [1,0.7,0.1,1],  
 radius = 0.5  
 )

Во кои ID-ата понатаму ке ги препратиме за создавање на Multibody.

Ги креираме со класата GravityMovement за полесна манипулација со нив:

planet = GravityMovement(mass, sphere3, visualShapeId, spherePos3, planetGravity)  
Neutron = GravityMovement(mass, sphere2, visualShapeId2, spherePos2, neutronGravity)  
Sun = GravityMovement(mass, sphere, visualShapeId3, spherePos1, sunGravity)

Во GravityMovement при иницијализација ги земаме атрибутите и дополнително пресметуваме и чуваме радиус на гравитација, позиција, тежина и ја креираме сферата со овие атрибути.

def \_\_init\_\_(self, mass, sphere, id, position, gravity):  
 self.position = position  
 self.gravity = gravity  
 self.gravityPosition = [x + self.gravity for x in position]  
 self.mass = mass  
 self.sphere = sphere  
 self.id = id  
 self.r\_hor = self.gravity  
 self.sphereUid = pybullet.createMultiBody(self.mass,  
 self.sphere,  
 self.id,  
 self.position,  
 )

Во класава ги имаме функциите за движење околу двата објекти испишани со координати:

def moveUp(self):  
 self.position = [self.position[0] - 0.005, self.position[1] + 0.002, self.position[2] + 0.005]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveCurvyUp(self):  
 self.position = [self.position[0] + 0.005, self.position[1] - 0.005, self.position[2] + 0.004]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveDown(self):  
 self.position = [self.position[0] + 0.006, self.position[1] + 0.00005, self.position[2] - 0.005]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveCurvyBack(self):  
 self.position = [self.position[0] - 0.005, self.position[1] - 0.001, self.position[2] + 0.004]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveCurvyDown(self):  
 self.position = [self.position[0] - 0.005, self.position[1] - 0.005, self.position[2] - 0.007]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
  
def moveCurvyForward(self):  
 self.position = [self.position[0] + 0.006, self.position[1] + 0.005, self.position[2] - 0.001]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveCurvyUpward(self):  
 self.position = [self.position[0] - 0.002, self.position[1] - 0.0085, self.position[2] - 0.001]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveCurvyBackwards(self):  
 self.position = [self.position[0] - 0.003, self.position[1] + 0.002, self.position[2] + 0.002]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveToDefault(self):  
 if not math.fabs(self.position[0]) < 0.004:  
 if self.position[0] < 0:  
 self.position[0] = self.position[0] + 0.004  
 elif self.position[0] > 0:  
 self.position[0] = self.position[0] - 0.004  
 else:  
 self.position[0] = 0  
 if not math.fabs(self.position[1]) < 0.008:  
 if self.position[1] < 0:  
 self.position[1] = self.position[1] + 0.008  
 elif self.position[1] > 0:  
 self.position[1] = self.position[1] - 0.008  
 else:  
 self.position[1] = 0  
 if not math.fabs(self.position[2]) < 0.0012:  
 if self.position[2] < 0:  
 self.position[2] = self.position[2] + 0.0012  
 elif self.position[2] > 0:  
 self.position[2] = self.position[2] - 0.0012  
 else:  
 self.position[2] = 0  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
  
def moveX(self):  
 self.position = [self.position[0] + 0.007, self.position[1] + 0.002, self.position[2] - 0.0001]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
  
def movecurvyAround(self):  
 self.position = [self.position[0] + 0.0031, self.position[1] - 0.006, self.position[2] - 0.002]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())  
def moveY(self):  
 self.position = [self.position[0] - 0.004, self.position[1], self.position[2] - 0.002]  
 self.setGrav(self.position)  
 pybullet.resetBasePositionAndOrientation(self.sphereUid, self.position, self.setRoll())

Дополнителна надворешна фукнција за пресметување на растојание:

def distance(centerPos1, centerPos2, r1, r2):  
 p1 = numpy.array(centerPos1); p2 = numpy.array(centerPos2)  
 sq\_dist = numpy.sum((p1 - p2)\*\*2, axis =0)  
 dist = numpy.sqrt(sq\_dist)  
 return dist - r1 - r2

Во главниот циклус – step симулацијата – имаме:

pybullet.stepSimulation()  
time.sleep(1./240.)

за да ја започне симулацијата.

Во главниот циклус се пресметува со flags каде се наоѓа во чиво гравитациско поле е моментално и Distance растојанието меѓу нив:

if neutronFlag == 1 and sunFlag == 0:  
 Distance = distance(planet.getCenter(), Neutron.getCenter(), planet.getGravityField(), Neutron.getGravityField())  
 if Distance > 0:  
 if planet.position[0] < 0 and planet.position[1] < 0 and planet.position[2] < 0:  
 planet.moveToDefault()  
 else:  
 planet.moveUp()  
 print("MOVING UP NOT YET IN GRAVITY")

понатаму има пресметки за кружење околу Neutron се додека не излезе од полето:

else:  
 print("CAUGHT GRAVITY")  
 if planet.position[2] <= Neutron.position[2] and planet.position[0] >= Neutron.position[0]:  
 print("MOVING CURVY UP")  
 planet.moveCurvyUp()  
 elif planet.position[0] >= Neutron.position[0]and planet.position[2] >= Neutron.position[2]:  
 print("MOVING CURVY BACK")  
 planet.moveCurvyBack()  
 elif planet.position[0] <= Neutron.position[0] and planet.position[2] >= Neutron.position[2]:  
 print("MOVING CURVY DOWN")  
 planet.moveCurvyDown()  
 elif planet.position[0] <= Neutron.position[0] and planet.position[2] <= Neutron.position[2]:  
 print("FINAL MOVEMENT BEFORE EXITING DOWN")  
 planet.moveDown()  
 neutronFlag = 0

Надвор од полето на Neutron:

if neutronFlag == 0 and sunFlag == 0:  
 print("EXITING GRAVITY")  
 if planet.position[0] <= Neutron.position[0] and planet.position[2] <= Neutron.position[2]:  
 print("STILL BEHIND THE NEUTRON ON THE WAY OUT")  
 planet.moveDown()  
 else:  
 print("ON ITS WAY OUT TO SUN GRAVITY")  
 planet.moveX()  
 sunFlag = 1

Кога ке влезе во Sun’s gravity се прават истите пресметки:

if sunFlag == 1 and neutronFlag == 0:  
 print("ON THE SUN'S GRAVITY")  
 Distance = distance(planet.getCenter(), Sun.getCenter(), planet.getGravityField(), Sun.getGravityField())  
 if Distance > 0:  
 print("NOT YET CAUGHT BY SUN")  
 planet.moveX()

Каде понатаму го има стандардното движење околу:

else:  
 if planet.position[2] >= Sun.position[2] and planet.position[0] <= Sun.position[0]:  
 print("MOVING CURVY FORWARD AROUND THE SUN")  
 planet.moveCurvyForward() #\*\*\*\*\*  
 elif planet.position[0] >= Sun.position[0] and planet.position[2] >= Sun.position[2]:  
 print("THE SWOOP")  
 planet.movecurvyAround() #\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 elif planet.position[0] >= Sun.position[0] and planet.position[2] <= Sun.position[2]:  
 print("MOVING AROUND BACK")  
 planet.moveCurvyUpward() # \*\*\*\*  
 if planet.position[1] < Sun.position[1]:  
 planet.moveY()  
 elif planet.position[2] <= Sun.position[2] and planet.position[0] <= Sun.position[0]:  
 print("MOVING UP BACKWARDS")  
 planet.moveCurvyBackwards()  
 # sunFlag = 0  
 neutronFlag = 1

На крај при излегување од полето на Sun се придвижува кон координатниот почеток назад:

if sunFlag == 1 and neutronFlag == 1:  
 if planet.position[2] < 0 or planet.position[0] > 0 or planet.position[1] < 0:  
 planet.moveToDefault()  
 else:  
 sunFlag = 0

За крај секогаш треба да се одјавиме од серверот:

pybullet.disconnect()