from math import pi, sin, cos

from panda3d.core import \*

from direct.task import Task

from blueprint import Blueprint as plan

import numpy as np

import random

from copy import deepcopy as dc

from direct.showbase.ShowBase import ShowBase

class Simulation(ShowBase):

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

#Setting up attributes i.e.FOV, Zoom, Color

ShowBase.\_\_init\_\_(self)

base.setBackgroundColor(0, 0, 0)

self.angularview = 0.0

zoompos = PerspectiveLens()

zoompos.setFov(50)

zoompos.setNear(0.01)

zoompos.setFar(10000)

base.cam.node().setLens(zoompos)

#Reading BP

blueprint = plan('test/floorplan\_1')

blueprint.read()

self.scene = blueprint.generateSimulation()

self.scene.reparentTo(self.render)

self.scene.setTwoSided(True)

#Creating Directional Light

lighttype1 = "dlight"

lightdirectional = DirectionalLight(lighttype1)

lightdirectional.setColor(VBase4(1, 1, 1, 1))

lightdirectionalnodes = self.render.attachNewNode(lightdirectional)

#Setting Nodes for lights

lightdirectionalnodes.setPos(0.5, 0.5, 3)

lightdirectionalnodes.lookAt(0.5, 0.5, 2)

self.render.setLight(lightdirectionalnodes)

#Setting up Positional Lights and Nodes

for i in range(10):

lighttype2 = "plight"

lightpositional = PointLight(lighttype2)

lightpositional.setAttenuation((1, 0, 1))

lightpositional.setColor(VBase4(10, 10, 10, 1))

lightpositionnodes = self.render.attachNewNode(lightpositional)

if i == 0:

lightpositionnodes.setPos(0.5, 0.5, 3)

self.render.setLight(lightpositionnodes)

#Introducing Ambient Light

lighttype = "alight"

self.lightambient = AmbientLight(lighttype)

self.lightambient.setColor(VBase4(0.2, 0.2, 0.2, 1))

#Ambient LightNodes

self.lightambientnodes = self.render.attachNewNode(self.lightambient)

self.render.setLight(self.lightambientnodes)

#Initiating Required Movement Function and View Changer (Front to Down)

base.disableMouse()

self.taskMgr.add(self.Keymovement, "Keymovement")

self.topview = [0.5, 0.5, 1.5]

self.requiredview = [0.5, 0.499, 0.5]

self.Heightadjust = 0

self.ogposition = blueprint.startCameraPos

self.ogposition2 = blueprint.startTarget

self.startH = 0

self.cam = self.topview

self.newlocation = self.requiredview

self.HeightAngle = self.Heightadjust

self.accept('enter', self.initiate)

self.mode = 'Front'

self.viewchanger = 1.02

ceiling = self.scene.find("\*\*/ceiling")

ceiling.hide()

return

#Function to initiate View change

def initiate(self):

self.viewchanger = 0

self.oldcam = dc(self.cam)

self.oldposition = dc(self.newlocation)

self.oldH = self.camera.getR()

if self.mode == 'Front':

self.newcam = self.ogposition

self.newposition = self.ogposition2

self.newH = self.startH

self.mode = 'Down'

else:

self.newcam = self.topview

self.newposition = self.requiredview

self.newH = self.Heightadjust

self.ogposition = dc(self.cam)

self.ogposition2 = dc(self.newlocation)

self.startH = self.camera.getR()

self.mode = 'Front'

pass

return

#Function to change view

def changing(self):

self.cam = []

self.newlocation = []

for c in range(3):

self.cam.append(self.oldcam[c] + (self.newcam[c] - self.oldcam[c]) \* self.viewchanger)

self.newlocation.append(self.oldposition[c] + (self.newposition[c] - self.oldposition[c]) \* self.viewchanger)

continue

self.HeightAngle = self.oldH + (self.newH - self.oldH) \* self.viewchanger

if self.viewchanger + 0.02 >= 1 and self.mode == 'Down':

ceiling = self.scene.find("\*\*/ceiling")

ceiling.show()

pass

if self.viewchanger <= 0.02 and self.mode == 'Front':

ceiling = self.scene.find("\*\*/ceiling")

ceiling.hide()

pass

return

# Function to assign and use keys

def Keymovement(self, task, mspeed = 0.003, rspeed = 0.01):

if self.viewchanger <= 1.01:

self.changing()

self.viewchanger += 0.02

pass

#Movement keys - WASD

if base.mouseWatcherNode.is\_button\_down('w'):

for c in range(2):

step = mspeed \* (self.newlocation[c] - self.cam[c])

self.cam[c] += step

self.newlocation[c] += step

continue

pass

if base.mouseWatcherNode.is\_button\_down('a'):

step = mspeed \* (self.newlocation[0] - self.cam[0])

self.cam[1] += step

self.newlocation[1] += step

step = mspeed \* (self.newlocation[1] - self.cam[1])

self.cam[0] -= step

self.newlocation[0] -= step

pass

if base.mouseWatcherNode.is\_button\_down('s'):

for c in range(2):

step = mspeed \* (self.newlocation[c] - self.cam[c])

self.cam[c] -= step

self.newlocation[c] -= step

continue

pass

if base.mouseWatcherNode.is\_button\_down('d'):

step = mspeed \* (self.newlocation[0] - self.cam[0])

self.cam[1] -= step

self.newlocation[1] -= step

step = mspeed \* (self.newlocation[1] - self.cam[1])

self.cam[0] += step

self.newlocation[0] += step

pass

#Rotation Keys - Left,Right,Up,Down

if base.mouseWatcherNode.is\_button\_down('arrow\_left'):

angularview = np.angle(complex(self.newlocation[0] - self.cam[0], self.newlocation[1] - self.cam[1]))

angularview += rspeed

self.newlocation[0] = self.cam[0] + np.cos(angularview)

self.newlocation[1] = self.cam[1] + np.sin(angularview)

pass

if base.mouseWatcherNode.is\_button\_down('arrow\_up'):

angularview = np.arcsin(self.newlocation[2] - self.cam[2])

angularview += rspeed

self.newlocation[2] = self.cam[2] + np.sin(angularview)

pass

if base.mouseWatcherNode.is\_button\_down('arrow\_right'):

angularview = np.angle(complex(self.newlocation[0] - self.cam[0], self.newlocation[1] - self.cam[1]))

angularview -= rspeed

self.newlocation[0] = self.cam[0] + np.cos(angularview)

self.newlocation[1] = self.cam[1] + np.sin(angularview)

pass

if base.mouseWatcherNode.is\_button\_down('arrow\_down'):

angularview = np.arcsin(self.newlocation[2] - self.cam[2])

angularview -= rspeed

self.newlocation[2] = self.cam[2] + np.sin(angularview)

pass

self.camera.setPos(self.cam[0], self.cam[1], self.cam[2])

self.camera.lookAt(self.newlocation[0], self.newlocation[1], self.newlocation[2])

self.camera.setR(self.HeightAngle)

return Task.cont

#Start Simulation

Simulation().run()