```
GlowScript 3.0 VPython
## MTH 182 Spring 2014-20202 Adam Hausknecht
## based on the hanoi.py program
## by ruth chabay, carnegie mellon university, 2000-06
## Revised for GlowScript 3.0 Summer 2020
## GlowScript 3.0 is much closer to Pyhon than
## previous versions of GlowScript
##

scene.width = 600
scene.height = 400
scene.userzoom = False
scene.center = vector(0, 2, 0)
scene.background = vector(0,1,1)
scene.range = 10

gFloor = box(pos = vector(0,-3.5,0), size = vector(23,.99,5),
             color = vector(1.0, 1.0, 1.0),
             texture = textures.wood,
             pickable =False)

modeLabel = label(pos = vector(0,-5.8,0), text = 'Autosolve!', box = False,
                  color = color.black, height = 24)

## Should make this a class for the Pegs
gPegY = -3
gPegHeight = 6
gPegRadi = 0.4
```

```
gPegTop = gPegHeight+gPegY
gMaxradius = 2.5
gPadding = 0.5
gSpacing = gMaxradius+gPadding

gLeftPeg = cylinder(pos = vector(-2*gSpacing, gPegY, 0),
                    size = vector(gPegHeight, gPegRadi, gPegRadi),
                    axis = vector(0,1,0), color = vector(0.8, 0.8, 0.8))

gMidPeg = cylinder(pos = vector(0, gPegY, 0),
                   size = vector(gPegHeight, gPegRadi, gPegRadi),
                   axis = vector(0,1,0), color = vector(0.8, 0.8, 0.8))

gRightPeg = cylinder(pos= vector(2*gSpacing, gPegY, 0),
                     size = vector(gPegHeight, gPegRadi, gPegRadi),
                     axis = vector(0,1,0))

gPegs = [gLeftPeg, gMidPeg, gRightPeg]
gPegDict = {"left": 0, "middle": 1, "right": 2}

gSrcPegIndex = 0; gSparePegIndex = 1; gDestPegIndex = 2
gPegStack = [[], [], []]


```

```
## Should make this a class for the Rings
gRings = [0, 0, 0, 0, 0]
gRingHues = [vector(1,0,0), vector(1,1,0), vector(0,1,0), vector(.3,.3,1), vector(1,0,1)]

for y in range(-2, 3):
    r = ring(pos = vector(-6, y, 0),
             radius = 1-y*.25,
             color = gRingHues[y+2],
             axis = vector(0,1,0),
             pickable = False)
    r.id = "Ring"+ (y+2).toString() ##Used for debuging
    gRings[y+2] = r

gNumberOfRings = 3

## Note integrated
class Rings():
    def __init__(self):
        self.numberOfRings = 3
        self.createRings()

    def createRings(self):
        rings = []
        ringHues = [vector(1,0,0), vector(1,1,0), vector(0,1,0), vector(.3,.3,1), vector(1,0,1)]
        for y in range(-2, 3):
             r = ring(pos = vector(-6, y, 0),
             radius = 1-y*.25,
             color = ringHues[y+2],
```

```
            axis = vector(0,1,0),
            pickable = False)
            r.id = "Ring"+ (y+2).toString()
##Used for debuging
            rings.append(r)
        self.rings = rings

    def setNumberOfRings(self, n):
        if (1 <= n) and (n <= 5):
            self.numberOfRings=n

    def number(self):
        return self.numberOfRings

    def moveDy(self, dy):
        for r in self.rings:
            r.pos.y += dy

    def resetY(self):
        for i, r in enumerate(self.rings):
            r.pos.y += i-2

    def resetRings(self, startingPegX, startingPegStack):
        for i, r in enumerate(self.rings):
            r.pos = vector(startingPegX, i-2,0)
            r.axis = vector(0,1,0)
            r.visible = True
            if i < self.numberOfRings:
                startingPegStack.push(r)
            else:
                r.visible = False
```

```
 ## Rings

#gPRings = Rings()


## Game state
gSolving = False
gMoves = 0
gAnimationRate = 25

class AutoSolveState():
    def __init__(self):
        self.fancySolveEnabled = False
        self.enabled = True; self.paused = False
        self.started = False; self.done = True
        self.depth = 0; self.stack = []
        self.pegNames = ["left", "middle", "right"]


    def reset(self, numberOfRings, srcPegIndex, destPegIndex, sparePegIndex):
        #alert("reset start")
        self.enabled = True; self.paused = False; self.started = False
        self.done = True; self.depth = 0; self.stack = []
        srcPeg = self.pegNames[gSrcPegIndex];
        destPeg = self.pegNames[gDestPegIndex];
        sparePeg = self.pegNames[gSparePegIndex];
        self.push(startStep, numberOfRings,
```

```
srcPeg, destPeg, sparePeg)
        #alert("reset: end")


    def push(self, step, nDisks, srcPeg, destPeg, sparePeg):
        self.stack.push([step, nDisks, srcPeg, destPeg, sparePeg])

    def pop(self): return self.stack.pop()

    def pendingMoves(self): return self.stack.length;

    def clearMoves(self): self.stack = []

    def pegName(index):
        if (0<= index) and (index < self.pegNames.length):
            return self.pegNames[index];
        else: return ""
## AutosolveState

class ManualSolveState():
    def __init__(self):
        self.enabled = False;
        self.started = False;
        self.paused = False;
        self.done = True;
        self.withAnimation = False;
        self.fancySolveEnabled = False;

```

```
    def reset(self, numberOfRings):
        self.enabled = True;
        self.started = False;
        self.paused = False;
        self.done = True;
        self.numberOfRings = numberOfRings;
## ManualSolveState


def doFinishFlash(lastMove, flashMax):

    solvePauseButton.text ="Done"
    if autosolveState.enabled:
        modeLabel.text = "Autosolve: Solved in "+ lastMove.toString() + " Moves!";
    elif manualsolveState.enabled:
        modeLabel.text = "Solve with Mouse: You Won in "+ lastMove.toString()+ " Moves!";

    rPeg = gPegDict['right']
    flash=0
    while flash < flashMax:
        gPegs[rPeg].color=color.red
        #gPRings.moveDy(0.5)
        for i in range(0, gNumberOfRings):
            gRings[i].pos.y += .5

        rate(10)
        gPegs[rPeg].color = color.white
        #gPRings.moveDy(0.5)
        for i in range(gNumberOfRings):
            gRings[i].pos.y -= .5
```

```

        rate(10)
        flash += 1

    gPegs[rPeg].color=color.white
    #gPRings.resetY()
    for i  in range(gNumberOfRings):
        gRings[i].pos.y = i-2

    setStateButtons(True)
## doFinishFlash

autosolveState = AutoSolveState()
manualsolveState = ManualSolveState()

## Interative solution contants
startStep = 1
returnStep = 4
animateStep = 5

def initStartingRingForAnimation(selectedRing, peg1, peg2):
    selectedRing.toPeg = peg2
    selectedRing.motion = "up"
    ##Set up the horizontal shift needed to reach the destPeg
    selectedRing.deltaX = gPegs[peg2].pos.x - gPegs[peg1].pos.x
    selectedRing.endingX = gPegs[peg2].pos.x

    ##Set up the vertical upward shift to remove the selected ring
```

```
    ##from the srcPeg
    selectedRing.deltaY = gPegTop + (selectedRing.size.y+selectedRing.size.x) - selectedRing.pos.y

    ## Set up the vertical downward shift to lower the selected ring
    ## to the top position on the destPeg
    selectedRing.endingY = -2 + gPegStack[peg2].length
## initStartingRingForAnimation

def startTopRingMove(srcPeg, destPeg, moveNumber):
    peg1 = gPegDict[srcPeg]; peg2 = gPegDict[destPeg]
    ##alert("startTopRingMove")
    selectedRing = gPegStack[peg1].pop()
    initStartingRingForAnimation(selectedRing, peg1, peg2)
    ##alert("after initStartingRingForAnimation")
    moveStr = moveNumber.toString()
    modeLabel.text = "Autosolve: Move "+moveStr
    return selectedRing
## startTopRingMove

def animateSelectedRingMove(selectedRing, trace):
    dx = 0.1; dy = 0.1; spinDelta = 10
    motion = selectedRing.motion

```

```
    if motion == "up":
        selectedRing.deltaY -= dy
        selectedRing.pos.y += dy
        ##Is the ring's upward motion finished? If so, move left or right
        if selectedRing.deltaY <= 0:
            selectedRing.pos.y = gPegTop + (selectedRing.size.y + selectedRing.size.x)
            if selectedRing.deltaX > 0:
                selectedRing.motion = "right"
            elif selectedRing.deltaX < 0:
                selectedRing.motion = "left"

    elif motion == 'left':
        selectedRing.deltaX += dx
        selectedRing.pos.x -= dx
        ## Is the ring's leftward motion finished? If so, move down or spin
        if selectedRing.deltaX >= 0:
            selectedRing.pos.x = selectedRing.endingX
            selectedRing.deltaY = selectedRing.pos.y-selectedRing.endingY
            if autosolveState.fancySolveEnabled:
                selectedRing.motion = 'spin'
                selectedRing.totalSpin = 0
            else:
               selectedRing.motion = "down"

    elif motion == "right":
        selectedRing.deltaX -= dx
        selectedRing.pos.x += dx
```

```
        ##Is the ring's rightward motion finished? If so, move down or spin
        if selectedRing.deltaX <= 0:
            selectedRing.pos.x = selectedRing.endingX;
            selectedRing.deltaY = selectedRing.pos.y-selectedRing.endingY;
            if autosolveState.fancySolveEnabled:
                selectedRing.motion = 'spin'
                selectedRing.totalSpin = 0
            else:
               selectedRing.motion = "down"

    elif motion == "down":
        selectedRing.deltaY -= dy
        selectedRing.pos.y -= dy
        ## Is the ring's downard motion finished? If so, add the ring to the
        ## top of the stack of rings on the "toPeg"
        if selectedRing.deltaY <= 0:
            selectedRing.motion = "finished"
            selectedRing.pos.y = selectedRing.endingY
gPegStack[selectedRing.toPeg].push(selectedRing)
            return returnStep

    elif motion == "spin":
        selectedRing.totalSpin += spinDelta
        ## Has the ring rotated 360 degrees; if so, then change the motion to downward.
```

```
        if selectedRing.totalSpin < 360:
            selectedRing.rotate(angle = radians(spinDelta), axis = vector(1,0,0), pos = vector(0,0,0))
        else:
            selectedRing.axis = vector(0,1,0)
            selectedRing.motion = "down"

    if trace:
       alert("| animating:"+selectedRing.motion)

    return animateStep
##animateSelectedRingMove

def recursiveCall(nDisks, srcPeg, destPeg, sparePeg, trace):
    if trace:
        alert("| End step 1" + startStep.toString() + ", " + nDisks.toString() + ", "+ srcPeg + ", " + destPeg+", " + sparePeg)
    return [startStep, nDisks, srcPeg, destPeg, sparePeg];
## recursiveCall

def doReturn(trace):
    values = autosolveState.pop()
    if trace:
      alert("| return: " +  values[0].toString() + ", " + values[1].toString() + ", "+ values[2] + ", " + values[3]+", " + values[4])
    return values;
## doReturn
```

```

def solveIteratively(trace):
    global gMove

    gMove = 0
    srcPeg = '?'; sparePeg ='?'; destPeg ='?'
    nDisks = 0; nextStep = 4; args = None

    if not autosolveState.started:
        autosolveState.started = True
        autosolveState.done = False
        if trace: alert("Tracing")

    while not autosolveState.done:
        setAnimationSpeed()
        if autosolveState.paused:
           modeLabel.text="Click Scene to Proceed"
           solvePauseButton.text ="Click Scene"
           scene.pause("")
           autosolveState.paused = False
           solvePauseButton.text ="Pause"

        rate(gAnimationRate)
        ## rotatePuzzle()
        if nDisks == 1:
            nDisks = 0; gMove += 1
            selectedRing = startTopRingMove(srcPeg, destPeg, gMove)
            nextStep = animateStep
            if trace:
              alert("| End step 0: " +
```

```
nextStep.toString()+", "+
                        nDisks.toString()+", "+srcPeg+", "+destPeg+", "+sparePeg)

        elif nextStep == 1:
            ## Simulate a Recursive Call: save return address and current values,
            ##                                pass new arguments
            autosolveState.push(nextStep + 1, nDisks, srcPeg, destPeg, sparePeg)
            args = recursiveCall(nDisks-1, srcPeg, sparePeg, destPeg, trace)
            nextStep = args[0]; nDisks = args[1]
            srcPeg = args[2]; destPeg = args[3]; sparePeg = args[4]
        elif nextStep == 2:
            ## Simulate a Recursive Call: save return address and current values,
            ##                                pass new arguments
            autosolveState.push(nextStep + 1, nDisks, srcPeg, destPeg, sparePeg)
            args = recursiveCall(1, srcPeg, destPeg, sparePeg, trace)
            nextStep = args[0]; nDisks = args[1]
            srcPeg = args[2]; destPeg = args[3]; sparePeg = args[4]
        elif nextStep == 3:
            ## Simulate a Recursive Call: save return address and current values,
            ##                                pass
```

```
new arguments
            autosolveState.push(nextStep+1, nDisks, srcPeg, destPeg, sparePeg)
            args = recursiveCall(nDisks-1, sparePeg, destPeg, srcPeg, trace)
            nextStep = args[0]; nDisks = args[1]
            srcPeg = args[2]; destPeg = args[3]; sparePeg = args[4]
        elif nextStep == returnStep:
            if autosolveState.pendingMoves():
                args = doReturn(False)
                nextStep = args[0]; nDisks = args[1]
                srcPeg = args[2]; destPeg = args[3]; sparePeg = args[4]
            else:
                autosolveState.done = True
                autosolveState.started = False
                doFinishFlash(gMove, 10)
                return

        elif nextStep == animateStep:
            nextStep = animateSelectedRingMove(selectedRing, trace)
## solveIteratively

def getSelectedPeg():
    obj = scene.mouse.pick();
    if obj == gPegs[0]:
        return 0
    elif obj == gPegs[1]:
        return 1
```

```
    elif obj == gPegs[2]:
        return 2
    else:
        return -1;
## getSelectedPeg

def getSrcPeg(moves, previousIllegal):

    if previousIllegal:
      modeLabel.text = str(moves) + ": Illegal Move! Click on the starting peg"
    else:
      modeLabel.text = str(moves) + ": Click on the starting peg"
    while not manualsolveState.done:
       scene.waitfor("click")
       peg = getSelectedPeg()
       if peg < 0:
           modeLabel.text ="No starting peg selected: Click again!"
       elif gPegStack[peg].length==0:
            modeLabel.text ="This peg is empty: Click another starting peg!"
       else:
           return peg
## getSrcPeg

def getDestPeg(moves, srcPeg, previousIllegal):

    if previousIllegal:
       modeLabel.text = str(moves)+ ": Illegal Move! Click another destination peg"
```

```
    else:
        modeLabel.text = str(moves)+ ": Click on the destination peg"

    while not manualsolveState.done:
        scene.waitfor("click")
        peg = getSelectedPeg()
        if peg < 0:
            modeLabel.text ="No destination peg selected: Click again!";
        elif gPegStack[peg].length==gNumberOfRings:
            modeLabel.text ="This peg is full: Click another destination peg!"
        elif peg==srcPeg:
            modeLabel.text = "The destination peg can't be the starting peg: \n"
            modeLabel.text += "Click another peg!"
        else:
            return peg
## getDestPeg

## Solve Manually
def solveWithMouse(trace):
    global gMoves

    peg1 = gSrcPegIndex; peg2 = -1; selectedRing = None
    illegalMove = False; illegalMoveCount = 0
    nextStep = -1

```

```
    if not manualsolveState.started:
        manualsolveState.started = True
        manualsolveState.done = False

    while not manualsolveState.done:
        setAnimationSpeed()
        if manualsolveState.paused:
            savedText = modeLabel.text
            solvePauseButton.text ="Click Scene"
            modeLabel.text = "Click Scene to Proceed"
            scene.pause("")
            manualsolveState.paused = False
            solvePauseButton.text ="Pause"
            modeLabel.text = savedText

        rate(gAnimationRate)
        if nextStep==animateStep:
            nextStep = animateSelectedRingMove(selectedRing, trace)
        else:
            if gPegStack[gDestPegIndex].length==manualsolveState.numberOfRings: ##task completed?

                doFinishFlash(gMoves, 10)
                manualsolveState.done = True
                return

            if peg1<0:
                peg1 = getSrcPeg(gMoves+1,
```

```
illegalMove)
                illegalMove = False; illegalMoveCount = 0

            if peg2<0:
                peg2 = getDestPeg(gMoves+1, peg1,illegalMove)

            illegalMove = False
            selectedRing=gPegStack[peg1].pop() ## remove top ring from starting peg
            len = gPegStack[peg2].length
            if len>0: ## stack not empty
                if gPegStack[peg2][len-1].size.y > selectedRing.size.y: ## legal move
                    gMoves += 1
                    if manualsolveState.withAnimation:
                        initStartingRingForAnimation(selectedRing, peg1, peg2)
                        nextStep = animateStep
                    else:
                        selectedRing.pos.x=gPegs[peg2].pos.x
                        selectedRing.pos.y=-2+len
                        gPegStack[peg2].push(selectedRing)
                    peg1 = -1; peg2 = -1
                else: ##illegal move
                    ## return the ring to the
```

```
starting peg

gPegStack[peg1].push(selectedRing)
                    peg2 = -1
                    illegalMove = True
                    illegalMoveCount += 1
                    if illegalMoveCount==2:
                        peg1 = -1
                        illegalMoveCount = 0

            else: ## stack empty
                gMoves += 1
                if manualsolveState.withAnimation:
                    initStartingRingForAnimation(selectedRing, peg1, peg2)
                    nextStep = animateStep
                else:
                    selectedRing.pos.x=gPegs[peg2].pos.x
                    selectedRing.pos.y=-2
                    gPegStack[peg2].push(selectedRing)
                peg1 = -1; peg2 = -1
## solveWithMouse


def setupPuzzle():
    x = gPegs[gSrcPegIndex].pos.x;
    for i in range(3):
```

```
        gPegStack[i] = []
    srcStack = gPegStack[gSrcPegIndex]

    for i in range(gRings.length):
        r = gRings[i];
        r.pos = vector(x, i-2,0)
        r.axis = vector(0,1,0)
        r.visible = True
        if i < gNumberOfRings:
           srcStack.push(r)
        else:
          r.visible = False
## setupPuzzle


def doReset():
    global gMoves, gSolving
    gMoves = 0
    gSolving = False
    if autosolveState.enabled:
        autosolveState.reset(gNumberOfRings, gSrcPegIndex, gDestPegIndex, gSparePegIndex)
        modeLabel.text = 'Autosolve!'
    elif manualsolveState.enabled:
        manualsolveState.reset(gNumberOfRings)
        modeLabel.text = 'Solve with Mouse! Click the Solve Button to Start!'

    setupPuzzle()
    setStateButtons(True)
    #alert("doReset: end")
## reset
```

```


def setFancySolve(enabled):
    autosolveState.fancySolveEnabled = enabled
    manualsolveState.fancySolveEnabled = enabled
##setFancySolve


def setAnimationSpeed():
    global gAnimationRate
    gAnimationRate = animationSpeedSlider.value
##setSpeed

def toggleSolveMode(choice):
    global gMoves, gSolving
    #global autosolveState
    #global modeLabel
    #global manualsolveState

    gMoves = 0
    gSolving = False
    if choice == 0:
        autosolveState.enabled = True
        autosolveState.paused = False
        modeLabel.text ='Autosolve!'
        manualsolveState.enabled = False
    else:
        autosolveState.enabled = False
        manualsolveState.enabled = True
        manualsolveState.paused = False
        manualsolveState.withAnimation =
(choice==1)
```

```
        modeLabel.text = 'Solve with Mouse! Click the Solve Button to Start!'

    setupPuzzle()
## toggleSolveMode


def setStartingPeg(startingPegMenu):
    global gSrcPegIndex, gSparePegIndex

    newStartingPeg = startingPegMenu.selected
    if (newStartingPeg=='Middle') and (gSrcPegIndex == gPegDict['left']):
        gSrcPegIndex = gPegDict['middle']
        gSparePegIndex = gPegDict['left']
        doReset()
    elif (newStartingPeg=='Left') and (gSrcPegIndex == gPegDict['middle']):
        gSrcPegIndex = gPegDict['left']
        gSparePegIndex = gPegDict['middle']
        doReset()
## setStartingPeg

def setNumberOfRings():
    global gNumberOfRings
    gNumberOfRings = int(nRingsMenu.selected)
    doReset()
##setNumberOfRings

def solve():
    global autosolveState, manualsolveState
    if autosolveState.enabled:
```

```
        if autosolveState.paused:
            autosolveState.paused = False
        else:
            doReset()
            doSolve()

    elif manualsolveState.enabled:
        if manualsolveState.paused:
            manualsolveState.paused = False;
        else:
            doReset()
            doSolve()
## solve


def doPause():
    global autosolveState, manualsolveState
    if autosolveState.enabled:
        autosolveState.paused = not autosolveState.paused
    elif manualsolveState.enabled:
        manualsolveState.paused = not manualsolveState.paused
## doPause


def doSolve():
    global gSolving
    gSolving = True
    setStateButtons(False)
    if autosolveState.enabled:
        solveIteratively(False)
```

```
    elif manualsolveState.enabled:
        solveWithMouse()
## doSolve


def changeMode(modeMenu):
    option = modeMenu.selected
    if option == "Autosolve":
        toggleSolveMode(0)
    elif option =="Manual + Animation":
        toggleSolveMode(1)
    else: ##Manual: No Animation:
        toggleSolveMode(2)
##changeMode


def doPauseSolve():
    if not gSolving:
        solvePauseButton.text = "Pause"
        solve();
    elif autosolveState.started or manualsolveState.started:
        doPause();
## doPauseSolve


def setStateButtons(enabled):
    modeMenu.disabled = not enabled
    startingPegMenu.disabled = not enabled
    nRingsMenu.disabled =  not enabled
##setStateButton

```

```
def handleResetButton():
    solvePauseButton.text = "Solve"
    doReset()


scene.title = "<span style='margin-right: 10px'><b>Towers Of Hanoi Puzzle</b></span>\n<b>Mode:</b>"
modeMenu = menu(choices=['Autosolve',
                         'Manual with Animation',
                         'Manual without Animation'], pos=scene.title_anchor,
                          index=0, bind=changeMode)

#scene.append_to_title("\n")
solvePauseButton = button(text='<b>Solve</b>', pos=scene.title_anchor, background=color.green, bind=doPauseSolve)

resetButton = button(text='<b>Reset</b>', background=color.red, pos=scene.title_anchor, bind=handleResetButton)

scene.append_to_title( "<span style='margin-left: 10px'>Start Peg:</span>")
startingPegMenu = menu(choices=[ 'Left', 'Middle'], pos=scene.title_anchor, index=0, bind=setStartingPeg)

scene.append_to_title("<span style='margin-left: 10px'>Rings:</span>")
```

```
nRingsMenu = menu(choices=['1', '2', '3', '4', '5'], pos=scene.title_anchor, index=2, bind=setNumberOfRings)

scene.caption = "Vary the animation rate: "
animationSpeedSlider = slider(min=10, max=200, value=25, length=220, bind=setAnimationSpeed, right=15)

def handleFancySolveCB(cb):
    setFancySolve(cb.checked)

checkbox(bind=setFancySolve, text='Fancy Solve')

setupPuzzle()
```