```
import maya.cmds as mc
import os
from functools import partial

'''
07_08_16
This is version 13 of Modelling as Animation, my first custom UI.
Last updated May 2013
'''


def makeJointSet(rButtonCollection):
    selectedJoints = mc.ls(selection=True)
    name = mc.textField("setName", query=True, text = True)
    fullName = "jointSet_" + '{name}'.format(name=name)
    print fullName

    jointSet = mc.sets(selectedJoints, name= fullName)
    #set the text field blank
    mc.textField("setName", edit=True, text ='name joint set')

    myButton = mc.radioButton(label=fullName,
collection=rButtonCollection)


#######        create animation Control Set
def makeAnimControlSet(anim_rButtonCollection):
    selectedControls = mc.ls(selection=True)
    name = mc.textField("controlSetName", query=True, text=True)
    fullName = "controlSet_" + '{name}'.format(name=name)
    print fullName
    controlSet = mc.sets(selectedControls, name= fullName)
    #set the text field blank
    mc.textField("controlSetName", edit=True, text ='name control set')
    myControlButton = mc.radioButton(label=fullName,
collection=anim_rButtonCollection)


#########      create animation control set radio buttons
def createAnimationRadioButtons(anim_rButtonCollection):
    #create radio button
    allSets = mc.ls(sets=True)
    print allSets
    myAnimSets = []
    for set in allSets:
        if "controlSet_" in set:
            myAnimSets.append(set)
    print myAnimSets
    for animSetName in myAnimSets:
        myControlButton = mc.radioButton(animSetName, label=animSetName,
collection=anim_rButtonCollection)


#######    query anim radio buttons
def queryAnimSetButton(*args):
    allSets = mc.ls(sets=True)
    myAnimSets = []
    for set in allSets:
```

```
        if "controlSet_" in set:
            myAnimSets.append(set)
    print myAnimSets
    for animSetName in myAnimSets:
        x = mc.radioButton(animSetName, query=True, select=True)
        if x == 1:
            animControlSet = animSetName
        else:
            pass
    return animControlSet

def createRadioButtons(rButtonCollection):
    #create radio button
    allSets = mc.ls(sets=True)
    print allSets
    mySets = []
    for set in allSets:
        if "jointSet_" in set:
            mySets.append(set)
    print mySets
    for setName in mySets:
        myButton = mc.radioButton(setName, label=setName,
collection=rButtonCollection)

def queryRadioButton(*args):
    allSets = mc.ls(sets=True)
    mySets = []
    for set in allSets:
        if "jointSet_" in set:
            mySets.append(set)
    print mySets
    for setName in mySets:
        x = mc.radioButton(setName, query=True, sl=True)
        if x == 1:
            activeJointSet = setName
        else:
            pass
    return activeJointSet


######  anim control button pressed
def cntrlButtonPressed(anim_rButtonCollection, *args):
    if not mc.ls(selection=True):
        print'Please select controls to create a control set'
    else:
        makeAnimControlSet(anim_rButtonCollection)
    #seems I need this 'launch()' in order to be able to select different
objects
    launchUI()


def buttonPressed(rButtonCollection,*args):
    if not mc.ls(selection=True):
        print'Please select joints to create a joint set'
    else:
        makeJointSet(rButtonCollection)
    #seems I need this 'launch()' in order to be able to select different
objects
```

```
    launchUI()


#####save and bind duplicate model

def duplicateModel():
    keySpacing = 4
    model = mc.ls(selection=True)
    #duplicate model
    duplicateModel = mc.duplicate(model)[0]
    #add duplicate to Models group
    mc.parent(duplicateModel, 'duplicateModels')
    #get current frame
    currentFrame = mc.currentTime( query=True )
    #set 0 vis at frame 1
    mc.setKeyframe( "{duplicate}.v".format(duplicate = duplicateModel),
time = 0, value = 0)
    firstKey = int(currentFrame - (keySpacing * 0.5))
    #set first keyframe
    mc.setKeyframe( "{duplicate}.v".format(duplicate = duplicateModel),
time=firstKey, value=1)
    secondKey = int(currentFrame + (keySpacing * 0.5))
    #set secondKey
    mc.setKeyframe( "{duplicate}.v".format(duplicate = duplicateModel),
time = secondKey, value = 0)
    #deselect model
    mc.select(clear=True)
    return duplicateModel



def saveAndBind(*args):
    #create duplicate model
    bindMesh = duplicateModel()
    print bindMesh
    bindJoints = queryRadioButton()
    print bindJoints
    mc.select(clear=True)
    mc.select(bindJoints)
    mc.select(bindMesh, add=True)
    mc.skinCluster(toSelectedBones=True, skinMethod=2,
maximumInfluences=3)


####  keyframe members of active control set
def keyControls(*args):
    keyCntrls = queryAnimSetButton()
    print keyCntrls
    mc.select(clear=True)
    mc.select(keyCntrls)
    mc.setKeyframe()
    mc.select(clear=True)



#####RIGGING
#create a set for each useful set of controls
```

```
#####SETUP
def layerAndGroup():
    #check for existence of models layer and add make it if not there
    layerTest = mc.objExists('duplicateModelslyr')
    if layerTest == 1:
        print 'duplicateModelslyr layer already exists'

    else:
        mc.createDisplayLayer(name="duplicateModelslyr", empty=True)
        print 'just made duplicateModelslyr layer'

    #check for existence of duplicateModels group and make it if not
there
    groupTest = mc.objExists('duplicateModels')
    if groupTest == 1:
        print 'duplicateModels group already exists'

    else:
        modelsGroup = mc.group(empty=True, name='duplicateModels')
        print 'I just made duplicatModels group'


    #check for existence of workingModels layer and add make it if not
there
    modelLayerTest = mc.objExists('workingModelslyr')
    if modelLayerTest == 1:
        print 'workingModels layer already exists'

    else:
        mc.createDisplayLayer(name="workingModelslyr", empty=True)
        print 'just made workingModelslyr layer'

    #check for existence of workingModel group and make it if not there
    modelGroupTest = mc.objExists('workingModel')
    if modelGroupTest == 1:
        print 'workingModel group already exists'

    else:
        modelGroup = mc.group(empty=True, name='workingModel')
        print 'I just made workingModel group'

    #add group to layer    Gina change this so that it doesnt add the
original model
    mc.editDisplayLayerMembers("duplicateModelslyr", "duplicateModels",
noRecurse=True)
    mc.setAttr('duplicateModelslyr.v', 0)
    mc.setAttr('duplicateModelslyr.color', 29)
    #add group to layer    Gina change this so that it doesnt add the
original model
    mc.editDisplayLayerMembers("workingModelslyr", "workingModel",
noRecurse=True)
    mc.setAttr('workingModelslyr.color', 8)


def createCube(*args):
    cube = mc.polyCube(name='workingModel', sh=2, sw=2, sd=2)
```

```
    #add cube to workingModel group
    mc.parent(cube, 'workingModel')


def deleteHistoryEtc(*args):
    mc.FreezeTransformations()
    mc.delete(all=True, constructionHistory=True)



def createShader(textureType='file', shaderType='lambert',
fileName='temp'):
    # create a shader
    refShader = mc.shadingNode(shaderType, asShader=True,
name='refShader1')
    # create a texture and set default colour
    refFile = mc.shadingNode(textureType, asTexture=True,
name='{fileName}'.format(fileName=fileName))        #
    mc.setAttr('{refFile}.defaultColor'.format(refFile=refFile), 1, 1, 1)
    #create and attach a placement node, set to no warp
    placementNode = mc.shadingNode("place2dTexture", asUtility=True)

mc.connectAttr('{placementNode}.outUV'.format(placementNode=placementNode
), '{refFile}.uvCoord'.format(refFile=refFile))

mc.connectAttr('{placementNode}.outUvFilterSize'.format(placementNode=pla
cementNode), '{refFile}.uvFilterSize'.format(refFile=refFile))

mc.setAttr('{placementNode}.wrapU'.format(placementNode=placementNode),
0)

mc.setAttr('{placementNode}.wrapV'.format(placementNode=placementNode),
0)
    # a shading group
    shadingGroup = mc.sets(renderable=True, noSurfaceShader=True,
empty=True)
    #connect shader to sg Lambert shader
    mc.connectAttr('{refShader}.outColor'.format(refShader=refShader),
'{shadingGroup}.surfaceShader'.format(shadingGroup=shadingGroup))
    #connect texture node to shader's color
    mc.connectAttr('{refFile}.outColor'.format(refFile=refFile),
'{refShader}.color'.format(refShader=refShader))
    return shadingGroup


def camAndRef(*args):
    #create camera
    modelCam = mc.createNode('camera')
    cameraTransform = mc.listRelatives(modelCam, allParents=True,
fullPath=True)
    mc.xform(cameraTransform, translation=(0,0,15))
    # find project directory
    projectDir = mc.workspace(query=True, rootDirectory=True)
    #convert it to the source images directory
    sourceImagesDir = projectDir + 'sourceimages/refPics'
    images = os.listdir(sourceImagesDir)

    for image in images:
```

```
        imageStr = str(image)
        imageName = imageStr.rpartition('.')[0]
        #create shader
        shader = createShader(fileName=str(imageName))
        #connect image to file node
        imagePath = sourceImagesDir+'/'+imageStr
        mc.setAttr(str(imageName)+'.fileTextureName', imagePath,
type='string')
        # create ref plane and parent under camera
        refPlane = mc.polyPlane(name='refPlane1', sw=1, sh=1, width=20,
height=20, axis=(0, 0, 1))[0]
        mc.xform(refPlane, translation=(0, 0, -50))
        # assign shader to plane
        mc.sets('{refPlane}'.format(refPlane=refPlane), e=1,
forceElement='{shader}'.format(shader=shader))
        # parent plane under camera
        mc.parent('{refPlane}'.format(refPlane=refPlane),
cameraTransform)


def setUp(*args):
    layerAndGroup()
    camAndRef()


# to set visibility keyframes on reference planes
def visibilityKeyframe(*args):
    currentObject = mc.ls(selection=True)[0]
    print 'currentObject   ', currentObject
    nodeType = mc.nodeType(currentObject)
    print nodeType

    if nodeType != 'transform':
        print 'not a transform node'
        currentObjectTransform = mc.listRelatives(currentObject,
allParents=True, allDescendents=True, type='transform')[0]
        print 'currentObjectTransform is   ' ,currentObjectTransform
    else:
        currentObjectTransform = currentObject
        print 'it was already a transform node'

    x =
mc.getAttr('{currentObjectTransform}.visibility'.format(currentObjectTran
sform=currentObjectTransform))
    if x<1:
        mc.setKeyframe(currentObjectTransform, attribute='visibility',
value=1)
    else:
        mc.setKeyframe(currentObjectTransform, attribute='visibility',
value=0)


#### create UI
def launchUI():
    #check to see if the window exists, create window
    if mc.window('myUI', query=True, exists=True):
        mc.deleteUI('myUI')
    # create the window
```

```
    WINDOW = mc.window('myUI', title='Modelling As Animation', w=300,
h=300)

    #create form layout
    form = mc.formLayout()
    #create tab Layout
    tabs = mc.tabLayout(innerMarginWidth=5, innerMarginHeight=5)
    #add tab layout to form layout
    mc.formLayout(form, edit=True, attachForm=((tabs, 'top', 0), (tabs,
'left', 0), (tabs, 'bottom', 0), (tabs, 'right', 0)))

    ##########       create tabOne - setup
    tabOne = mc.columnLayout('setup', adjustableColumn=True,
backgroundColor=(0.7, 0.8, 0.9))
    mc.separator(h=40, parent=tabOne)
    mc.text("1. put your reference images in 'sourceimages\\ refPics'
folder")
    mc.separator(h=40, parent=tabOne)
    mc.text(label='2. set the maya project directory')
    projectButton = mc.button(label='SET PROJECT',
command=(mc.SetProject), backgroundColor=(0.9, 0.9, 1), h=45)
    mc.separator(h=40, parent=tabOne)
    mc.text(label='3. create camera, reference planes, group and display
layer')
    camButton = mc.button(label='CREATE CAMERA etc', command=(setUp),
backgroundColor=(0.9, 0.9, 1), h=45)
    mc.separator(h=40, parent=tabOne)
    mc.text(label='4. animate visibity of the reference planes')
    refVisibilityButton = mc.button(label='KEY VISIBILITY',
command=(visibilityKeyframe), backgroundColor=(0.9, 0.9, 1), h=45)

    mc.separator(h=40, parent=tabOne)
    mc.text(label='5. create low poly model(s)', parent=tabOne)
    cubeButton = mc.button(label='CREATE CUBE', command=(createCube),
backgroundColor=(0.9, 0.9, 1), h=45)
    mc.separator(h=40, parent=tabOne)
    mc.text(label='6. name the model, delete history and freeze
transforms', parent=tabOne)
    cubeButton = mc.button(label='DELETE HISTORY etc',
command=(deleteHistoryEtc), backgroundColor=(0.9, 0.9, 1), h=45)
    mc.setParent(tabs)

    ###########      create tabTwo - rigging and animation
    tabTwo = mc.columnLayout('rigging and animation',
adjustableColumn=True, backgroundColor=(0.7, 0.8, 0.9))
    mc.separator(h=40, parent=tabTwo)
    mc.text(label='1. create joints and controls', parent=tabTwo)
    jointToolButton = mc.button(label='JOINT TOOL',
command=(mc.JointTool), backgroundColor=(0.9, 0.9, 1), h=45,
parent=tabTwo)
    mc.separator(h=40, parent=tabTwo)
    mc.text(label='2. bind model to joints', parent=tabTwo)
    bindlButton = mc.button(label='SMOOTH BIND TO ALL JOINTS',
command=('mc.skinCluster(toSelectedBones=False)'), backgroundColor=(0.9,
0.9, 1), h=45, parent=tabTwo)
    mc.separator(h=40, parent=tabTwo)
    mc.text(label='3. create sets of animation controls to keyframe',
parent=tabTwo)
```

```
    #create radio button collection
    anim_rButtonCollection = mc.radioCollection()
    cntrlTextWindow = mc.textField("controlSetName", width=100,
insertText='name the animation control set')
    #make the create set button
    createCntrlSetButton = mc.button(command=(partial(cntrlButtonPressed,
anim_rButtonCollection)), label='ADD CONTROL SET', backgroundColor=(0.9,
0.9, 1), h=45)
    mc.text(label='active control set ...', parent=tabTwo)
    #create initial radio buttons for existing animation control sets in
the scene
    createAnimationRadioButtons(anim_rButtonCollection)
    mc.separator(h=40, parent=tabTwo)
    mc.text(label='4. set keys on active controls', parent=tabTwo)
    keyControlsButton = mc.button(command=(keyControls), label='SET
KEYS', backgroundColor=(1, 0.5, 0.5), h=45)
    mc.setParent(tabs)

    #############     create tabThree - modelling
    tabThree = mc.columnLayout('modelling and animation', rowSpacing=5,
adjustableColumn=True, backgroundColor=(0.7, 0.8, 0.9))
    mc.separator(h=40, parent=tabThree)
    mc.text(label='1. create joint sets to which duplicate model will be
bound', parent=tabThree)
    #create radio button collection
    rButtonCollection = mc.radioCollection(parent=tabThree)
    textWindow = mc.textField("setName", width=100, insertText='name the
joint set')
    #make the create set button
    createSetButton = mc.button(command=(partial(buttonPressed,
rButtonCollection)), label='ADD JOINT SET', backgroundColor=(0.9, 0.9,
1), h=45)
    mc.text(label='active joint set ...', parent=tabThree)
    #create initial radio buttons for existing sets in the scene
    createRadioButtons(rButtonCollection)

    # key spacing option
    mc.separator(h=40, parent=tabThree)
    mc.text(label='key spacing = 4', parent=tabThree)

    # delete history option
    mc.separator(h=40, parent=tabThree)
    mc.text(label='delete history before bind = 0', parent=tabThree)

    # duplicate model
    mc.separator(h=40, parent=tabThree)
    mc.text(label='duplicate, bind and keframe new model',
parent=tabThree)
    saveAndBindButton = mc.button(command=(saveAndBind), label='DUPLICATE
MODEL', backgroundColor=(1, 0.5, 0.5), h=45)

    #step forward and step back(based on keyspacing)

    mc.setParent(tabs)
    mc.showWindow(WINDOW)

launchUI()
```