```
#! /usr/bin/env python

from taskHandler import Location, Task, TaskFactory
import roslib; roslib.load_manifest('smart_stool')
import rospy
from geometry_msgs.msg import PoseStamped, Twist, Vector3
from nav_msgs.msg import Odometry
from kobuki_msgs.msg import BumperEvent
from move_base_msgs.msg import MoveBaseActionResult
from tf.transformations import quaternion_about_axis, euler_from_quaternion
z_axis = (0,0,1)
from math import pi

class SmartStool:
    def __init__(self):
        # state of the smart stool
        self.odomPose = Location(0,0,0)
        self.bumperTriggered = False
        self.atTaskLocation = False

        # defining the tasks
        stool = Task('stool', 1, Location(0,0,0), 'sit')
        getMail = Task('get_mail', 2, Location(4,-3,0), 'bump')
        chasePets = Task('chase_pets', 3, Location(0,0,0), 'wiggle')
        charge = Task('charge_battery', 4, Location(1,0,0), 'sit')
        charge.activate() # charging should always be an active task

        # populate the task list and set up the task factory
        taskList = [stool, getMail, chasePets, charge]
        self.factory = TaskFactory(taskList)

        # set up the current task
        self.task = self.factory.getNextTask()

        # set up the subscribers
        self.odom_sub = rospy.Subscriber('/odom', Odometry, self.readOdometry, queue_size=1)
        self.bumper_sub = rospy.Subscriber('/mobile_base/events/bumper', BumperEvent,
self.readBumper, queue_size=1)
        self.goalReached_sub = rospy.Subscriber('/move_base/result', MoveBaseActionResult,
self.goalReached, queue_size=1)

        # set up the publishers
        self.moveBase_pub = rospy.Publisher('/move_base_simple/goal', PoseStamped)
        self.action_pub = rospy.Publisher('/cmd_vel_mux/input/teleop', Twist)

    def goToTask(self):
        # send the smart stool to the location of its current task
        current_task_location = self.task.location.copy()
        goal = PoseStamped()
```

```
        goal.header.frame_id = 'map'
        goal.header.seq = 1
        now = rospy.Time.now()
        goal.header.stamp.secs = now.secs
        goal.header.stamp.nsecs = now.nsecs
        goal.pose.position.x = current_task_location.x
        goal.pose.position.y = current_task_location.y
        goal.pose.position.z = 0
        quat = quaternion_about_axis(current_task_location.theta,z_axis)
        goal.pose.orientation.w = quat[0]
        goal.pose.orientation.x = quat[1]
        goal.pose.orientation.y = quat[2]
        goal.pose.orientation.z = quat[3]
        self.moveBase_pub.publish(goal)

    def publishTwist(self, cmd_linvel, cmd_angvel):
        # publishes a Twist message to /cmd_vel_mux/input/teleop to perform custom motion actions
        self.action_pub.publish(Twist(Vector3(cmd_linvel,0,0),Vector3(0,0,cmd_angvel)))

    def actionHandler(self,actionName):
        ####
        #### TODO: a change of task priority doesn't necessarily mean that the task was deactivated.
Need to check
        #### if original task is still in list of active tasks. if it is, do not deactivate it. if it's not, deactivate
it.
        #### Also need to check for other general silly mistakes
        ####
        current_task = self.task.copy()
        startLocation = self.odomPose.copy()
        driveSpeed = 0.1
        spinSpeed = 0.5
        close_enough = 0.1
        wiggle_rotate = pi/2
        timeout = 10
        startTime = rospy.get_time()

        # execute the sit action
        print actionName
        if actionName == 'sit':
            while (not rospy.is_shutdown()) and (self.task == current_task):
                self.publishTwist(0,0)
                rate.sleep()
                self.task = self.factory.getNextTask()
                ##### TEMP #####
                self.factory.activateTask('get_mail')

        # execute the bump action
        elif actionName == 'bump':
            self.bumperTriggered = False
```

```
            while not rospy.is_shutdown() and not self.bumperTriggered:
                self.publishTwist(driveSpeed,0)
                rate.sleep()
            startTime = rospy.get_time()
            while not rospy.is_shutdown() and (rospy.get_time() - startTime < 1):
                self.publishTwist(-driveSpeed,0)
                rate.sleep()
            self.factory.deactivateTask(current_task.name)

        # execute the wiggle action
        elif actionName == 'wiggle':
            while self.task == current_task or (rospy.get_time() - startTime > timeout):
                while not rospy.is_shutdown() and not self.odomPose.compareAngle(startLocation,-
wiggle_rotate):
                    self.publishTwist(0,-spinSpeed)
                    rate.sleep()
                while not rospy.is_shutdown() and not
self.odomPose.compareAngle(startLocation,wiggle_rotate):
                    self.publishTwist(0,spinSpeed)
                    rate.sleep()
                self.task = self.factory.getNextTask()
            self.factory.deactivateTask(current_task.name)

        # warn that the specified action is not implemented
        else:
            print 'Action not implemented!'
            print actionName

        # stop the robot:
        self.publishTwist(0,0)

    def execute(self):
        if self.task is None: break
        current_task = self.task.copy()
        self.goToTask()
        # wait for the robot to be at its goal position
        print 'going to task:' + current_task.name
        while not self.atTaskLocation:
            rate.sleep()
            self.task = self.factory.getNextTask()
            # if that task has changed, exit this function
            if not(current_task == self.task):
                return
        # reset for the next task
        self.atTaskLocation = False
        print 'doing action'
        self.actionHandler(self.task.getAction())

    def readOdometry(self,msg):
```

```
        # callback function to read the robot's current odometry position
        odom_position = msg.pose.pose.position
        odom_rotation = msg.pose.pose.orientation
        self.odomPose =
Location(odom_position.x,odom_position.y,euler_from_quaternion((odom_rotation.w,
odom_rotation.x, odom_rotation.y, odom_rotation.z))[2])

    def readBumper(self,msg):
        # callback function to set the bumperTriggered flag if the bumper was hit
        self.bumperTriggered = True

    def goalReached(self,msg):
        # callback function to determine if the current task location was reached
        if msg.status.status == 3:
            self.atTaskLocation = True

if __name__ == '__main__':
    # initialize the node:
    rospy.init_node('smart_stool')
    freq = 30 # hz
    rate = rospy.Rate(freq)
    # set up the smart stool object
    mySmartStool = SmartStool()

    # wait for one second
    for i in range(freq):
        rate.sleep()

    while not rospy.is_shutdown():
        mySmartStool.execute()
        rate.sleep()



#top = factory.getNextTask()
#all = factory.getAllTasks()
```