```
import rclpy
from rclpy.node import Node
import math
import random
from rclpy.parameter import Parameter
from rcl_interfaces.msg import ParameterDescriptor
import turtle

from geometry_msgs.msg import Twist, Pose

from turtle_interfaces.srv import SetColor
from turtle_interfaces.msg import TurtleMsg

class TurtleClient(Node):
    def __init__(self):
        super().__init__('turtleClient')

        #### Display/Turtle Setup ####
        self.screen = turtle.Screen()
        self.screen.bgcolor('lightblue')
        self.turtle_display = turtle.Turtle()
        self.turtle_display.shape("turtle")
        self.turtle = TurtleMsg()

        #### publisher define ####
        self.twist_pub = self.create_publisher(Twist, 'turtleDrive', 1)
        ############################

        self.turtle_sub = self.create_subscription(TurtleMsg, 'turtleState', self.turtle_callback, 1)
        self.declare_parameter('turtleColor', 'blue', ParameterDescriptor(description='Default color
of the turtle'))

        #Keegan Cazalet added the ability to change the color of the turtle on 2/22/25
        turtleColor = self.get_parameter('turtleColor').get_parameter_value().string_value
        self.turtle_display.color(turtleColor)
        self.color_cli = self.create_client(SetColor,'SetColor')

        while not self.color_cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('Color service not available, waiting...')

        self.color_req = SetColor.Request()
        self.color_req.color = self.get_parameter('turtleColor').get_parameter_value().string_value
        self.server_call = True
        self.service_future = self.color_cli.call_async(self.color_req)
```

```
        #Keegan Cazalet created a parameter in which sets the pen size on 2/27/25
        self.declare_parameter('penSize', 2,  ParameterDescriptor(description='Default pen size for
the turtle'))
        pen_size = self.get_parameter('penSize').get_parameter_value().integer_value
        self.turtle_display.pensize(pen_size)

    def turtle_callback(self, msg):

        self.turtle = msg

    def update(self):

        if self.turtle.color == 'None':
            self.turtle_display.penup()
        else:
            self.turtle_display.pencolor(self.turtle.color)

        self.turtle_display.setpos(self.turtle.turtle_pose.position.x, self.turtle.turtle_pose.position.y)

        roll, pitch, yaw = rpy_from_quat(self.turtle.turtle_pose.orientation.x,
                                         self.turtle.turtle_pose.orientation.y,
                                         self.turtle.turtle_pose.orientation.z,
                                         self.turtle.turtle_pose.orientation.w)
        self.turtle_display.seth(math.degrees(yaw))

def quat_from_rpy(roll, pitch, yaw):

    cy = math.cos(yaw*0.5)
    sy = math.sin(yaw*0.5)
    cp = math.cos(pitch*0.5)
    sp = math.sin(pitch*0.5)
    cr = math.cos(roll*0.5)
    sr = math.sin(roll*0.5)

    qw = cr * cp * cy + sr * sp * sy
    qx = sr * cp * cy - cr * sp * sy
    qy = cr * sp * cy + sr * cp * sy
    qz = cr * cp * sy - sr * sp * cy

    return qx, qy, qz, qw

def rpy_from_quat(x, y, z, w):
```

```
    srcp = 2*(w*x + y*z)
    crcp = 1-2*(x*x + y*y)
    roll = math.atan2(srcp, crcp)

    sp = 2*(w*y - z*x)
    if math.fabs(sp) >= 1:
        pitch = (sp/math.fabs(sp))*math.pi/2
    else:
        pitch = math.asin(sp)

    sycp = 2*(w*z + x*y)
    cycp = 1 - 2*(y*y + z*z)
    yaw = math.atan2(sycp, cycp)

    return roll, pitch, yaw

def main(args=None):

    #initial ROS2
    rclpy.init(args=args)

    #initial turtle client
    cli_obj = TurtleClient()
    cli_obj.get_logger().info('Turtlebot Client Started!')

    while rclpy.ok():

        cli_obj.update()
        rclpy.spin_once(cli_obj)

        unit_x = 1 #<put a reasonable ratio, 1 is a good number, around 1 is good enough>
        unit_z = 1 #<put a reasonable ratio, 1 is a good number, around 1 is good enough>


        cmd_msg = Twist()
        cmd_msg.linear.x = float(50 * unit_x)
        cmd_msg.angular.z = float(1 * unit_z)
        #Keegan Cazalet commented out publishing the command velocity on 2/21/2025
        #cli_obj.twist_pub.publish(cmd_msg)


    # Destory the node explicitly
    cli_obj.destroy_node()
    rclpy.shutdown()
```

```
if __name__=='__main__':
    main()
```