Code Listing

Typeset using LaTeX.

Tele-Op:

```
//NOTE: table thingy up = 0.5 and down = 0.2

package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.*;

import org.firstinspires.ftc.teamcode.Util;

import static org.firstinspires.ftc.teamcode.RelicRecovery.MecanumTeleop.glyphIntakeStatus.
    ↪ HASTWO;
import static org.firstinspires.ftc.teamcode.RelicRecovery.MecanumTeleop.glyphIntakeStatus.
    ↪ INTAKEOFF;
import static org.firstinspires.ftc.teamcode.RelicRecovery.MecanumTeleop.glyphIntakeStatus.
    ↪ SERVOUP;


@TeleOp(name = "TeleOp", group = "TeleOp")
//@Disabled
public class MecanumTeleop extends OpMode {

    DcMotor rightFront, leftFront, leftBack, rightBack;
    DcMotor intake1, intake2;
    DcMotor trayLift;
    DcMotor relic;


    Servo rightTilt, leftTilt;
    Servo JewelArmServo;
    Servo JewelWhackerServo;
    Servo relicArmServo, relicGrabberServo;
    Servo tableBackstopServo;

    //DigitalChannel trayTouch; //future touchsensor

    double ScalingInitialPower = 0.05;
    double GamepadDeadZone = 0.05;
    double FAST_MAX_POWER = 1;
    double SLOW_MAX_POWER = 0.8;
    double MAX_POWER;
    boolean FastGear = true;
    boolean triggerCanbeRead = true;

    double VelocityLeftFront;
    double VelocityRightFront;
    double VelocityLeftBack;
    double VelocityRightBack;

    int intakeStatus = 0;
    boolean intakeChanged = false;
    private final double INTAKE_POWER = 0.95;
```

```
    boolean dPadWasPressed = false;

    boolean canTurnIntakeOff = true;

//    double initialLeftStickY;
//    double initialLeftStickX;
//    double initialRightStickX;

    double leftStickY;
    double leftStickX;
    double rightStickX;

    double G2leftStickY;

    //tray conditions for new tray
    boolean tilitingTray = false;
    boolean raisingTray = true;

    int untiltedPos = -5; //giving margin at the bottom
    int tiltedPos = -470;
    int liftUp = -1525;
    int liftDown = -70; //giving margin at the bottom
    int horizontalPos = -60;

    boolean traySensor = false;
    boolean presets = false, grippersClosed = false;
    long lastbuttontime = System.nanoTime();

    // tilting up means dumping the glyph
    boolean trayCanGoUp = true, trayCanGoDown = false, trayCanTiltUp = true, trayCanTiltDown
        ↪ = false;

    boolean rightStickActive = false, leftStickActive = false;

    boolean aButtonIsPressed = false;
    boolean bButtonIsPressed = false;
    boolean yButtonIsPressed = false;
    boolean xButtonIsPressed = false;

    boolean aButtonCanBePressed = true;
    boolean bButtonCanBePressed = true;
    boolean yButtonCanBePressed = true;
    boolean xButtonCanBePressed = true;

    boolean topPreset = false;
    boolean bottomPreset = false;
    boolean tiltPreset = false;

    //whacker things
    static double JewelArmServoUp = 0.85;
    static double JewelWhackerServoUp = 0.9;

    //Swich to relic mode by holding both bumpers

    boolean RelicMode = false;
    boolean pad2bumpersCanbeRead = true;
```

```
boolean G2DpadUPIsPressed = false;
boolean G2DpadDOWNIsPressed = false;

boolean G2DpadUPCanBePressed = true;
boolean G2DpadDOWNCanBePressed = false;

boolean relicCanExtend = true;
boolean relicCanRetract = false;

boolean ExtendingRelic = true;


double RelicArmUp = 0.81;
double RelicArmDown = 0.26;
double RelicArmNearDown = 0.28;
double GrabberClosed = 0.53;
double GrabberOpen = 0.31;

double tableBackstopServoDown = 0.57;
double tableBackstopServoUp = 0.73;

double currentRelicArmPos;

static DistanceSensor glyphCounterDistance;
static com.qualcomm.robotcore.hardware.ColorSensor glyphCounterColor;

enum glyphIntakeStatus {
    HASTWO, INTAKEOFF, SERVOUP
}

static glyphIntakeStatus glyphCountStatus;

long initialTime;
long elapsedTime;

int counter = 0;

public void init() {

    trayLift = hardwareMap.dcMotor.get("lift");
    trayLift.setMode(DcMotor.RunMode.STOP_AND_RESET_ENCODER);

    relic = hardwareMap.dcMotor.get("relic");
    relicArmServo = hardwareMap.servo.get("relicArm");
    relicArmServo.setPosition(0.07);
    relicGrabberServo = hardwareMap.servo.get("relicGrabber");
    relicGrabberServo.setPosition(GrabberClosed);

    rightBack = hardwareMap.dcMotor.get("rightBack");
    leftBack = hardwareMap.dcMotor.get("leftBack");
    rightFront = hardwareMap.dcMotor.get("rightFront");
    leftFront = hardwareMap.dcMotor.get("leftFront");

    intake1 = hardwareMap.dcMotor.get("intake_left");
    intake2 = hardwareMap.dcMotor.get("intake_right");
```

```
        rightBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        leftBack.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        rightFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        leftFront.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        // trayTouch = hardwareMap.digitalChannel.get("trayTouch");
        //TODO:set init positions for table
        rightTilt = hardwareMap.servo.get("rightTiltServo");
        leftTilt = hardwareMap.servo.get("leftTiltServo");
//        leftTilt.setPosition(LeftServoDegreesToServoPos(-16));
//        rightTilt.setPosition(RightServoDegreesToServoPos(-16));

        tableBackstopServo = hardwareMap.servo.get("tableBackstopServo");
        tableBackstopServo.setPosition(tableBackstopServoDown);

        JewelArmServo = hardwareMap.servo.get("JewelArmServo");
        JewelArmServo.setPosition(JewelArmServoUp);
        JewelWhackerServo = hardwareMap.servo.get("JewelWhackerServo");
        JewelWhackerServo.setPosition(JewelWhackerServoUp);


        rightBack.setDirection(DcMotor.Direction.REVERSE);
        rightFront.setDirection(DcMotor.Direction.REVERSE);

        trayLift.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);


        trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);

        glyphCounterColor = hardwareMap.colorSensor.get("glyphCounterDistanceColor");
        glyphCounterDistance = hardwareMap.get(DistanceSensor.class,"
            ↪ glyphCounterDistanceColor");

        // eventually calls code that throws Null Pointer
        // added solution: initialize motorsWithEncoders array
        // TODO: make sure these motors are appropriate for Tele-Op
        Util.motorsWithEncoders = new DcMotor[]{rightFront, rightBack, leftFront, leftBack,
            ↪ trayLift, relic};
        IntakeControl.teleOpinit();


        initialTime = 0;
        elapsedTime = 0;

        glyphCountStatus = HASTWO;

        getGlyphs.glyphCount = 0;

//        rightFront.setMode(DcMotor.RunMode.RESET_ENCODERS);
//        leftFront.setMode(DcMotor.RunMode.RESET_ENCODERS);
//        rightBack.setMode(DcMotor.RunMode.RESET_ENCODERS);
//        leftBack.setMode(DcMotor.RunMode.RESET_ENCODERS);
    }

    public double ScalingFunctionPositive(double x) {
        double y = (Math.pow(x, 2) - (0.05 * x) + ScalingInitialPower);
        return y;
```

```
    }

    public double ScalingFunctionNegative(double x) {
        double y = ((-(Math.pow(x, 2))) - (0.05 * x) - ScalingInitialPower);
        return y;
    }

    public double ScalingFunctionPositiveCubic(double x) {
        double y = (Math.pow(x, 3) - (0.05 * x) + 0.05);
        return y;
    }

    public double ScalingFunctionNegativeCubic(double x) {
        double y = (Math.pow(x, 3) - (0.05 * x) - 0.05);
        return y;
    }

    public static double RightServoDegreesToServoPos(double degrees) {
        double y = (((4.8* Math.pow(10, -3)) * degrees) + 0.368);
        return y;
    }

    public static double LeftServoDegreesToServoPos(double degrees) {
        double y = (((-4.8* Math.pow(10, -3)) * degrees) + 0.672);
        return y;
    }

    public void loop() {

        useDPad();

        if ((gamepad1.right_trigger > 0.3 || gamepad1.left_trigger > 0.3) & triggerCanbeRead)
            ↪  {
            FastGear = !FastGear;
            triggerCanbeRead = false;
        }
        if (!triggerCanbeRead & (gamepad1.right_trigger < 0.2 & gamepad1.left_trigger < 0.2))
            ↪  {
            triggerCanbeRead = true;
        }

        if (!dPadWasPressed) {
            if (FastGear) {
                MAX_POWER = FAST_MAX_POWER;
            } else {
                MAX_POWER = SLOW_MAX_POWER; //may have to increase slow max power if continue
                    ↪  to use cubic
            }

            leftStickY = -gamepad1.left_stick_y * MAX_POWER;
            leftStickX = gamepad1.left_stick_x * MAX_POWER;
            rightStickX = gamepad1.right_stick_x * MAX_POWER * 0.8; //lowered rotation speed
                ↪ - needs testing

//            initialLeftStickY = leftStickY;
//            initialLeftStickX = leftStickX;
//            initialRightStickX = rightStickX;
```

```
//            if ((Math.abs(-gamepad1.left_stick_y) - Math.abs(initialLeftStickY)) > 0.3) {
//                leftStickY = -gamepad1.left_stick_y * 0.5;
//                initialLeftStickY *= 2;



            if ((leftStickY >= -GamepadDeadZone) & (leftStickY <= GamepadDeadZone)) { //if
                ↪ stick is in dead zone set value equal to zero
                leftStickY = 0;
            } else if ((leftStickY >= GamepadDeadZone) & (leftStickY <= 1)) { //if stick is
                ↪ pos use scaling with pos y-int
//                leftStickY = ScalingFunctionPositive(leftStickY);
                leftStickY = ScalingFunctionPositiveCubic(leftStickY);
            } else if ((leftStickY <= -GamepadDeadZone) & (leftStickY >= -1)) { //if stick is
                ↪  neg us scaling with neg y-int
                //leftStickY = ScalingFunctionNegative(leftStickY);
                leftStickY = ScalingFunctionNegativeCubic(leftStickY);
            }

            if ((leftStickX >= -GamepadDeadZone) & (leftStickX <= GamepadDeadZone)) {
                leftStickX = 0;
            } else if ((leftStickX >= GamepadDeadZone) & (leftStickX <= 1)) {
                //leftStickX = ScalingFunctionPositive(leftStickX);
                leftStickX = ScalingFunctionPositiveCubic(leftStickX);
            } else if ((leftStickX <= -GamepadDeadZone) & (leftStickX >= -1)) {
                //leftStickX = ScalingFunctionNegative(leftStickX);
                leftStickX = ScalingFunctionNegativeCubic(leftStickX);

            }

            if ((rightStickX >= -GamepadDeadZone) & (rightStickX <= GamepadDeadZone)) {
                rightStickX = 0;
            } else if ((rightStickX >= GamepadDeadZone) & (rightStickX <= 1)) {
                //rightStickX = ScalingFunctionPositive(rightStickX);
                rightStickX = ScalingFunctionPositiveCubic(rightStickX);
            } else if ((rightStickX <= -GamepadDeadZone) & (rightStickX >= -1)) {
//                rightStickX = ScalingFunctionNegative(rightStickX);
                rightStickX = ScalingFunctionNegativeCubic(rightStickX);
            }



        /*
        if the left stick is in the middle and the right stick is not rotate

        if the left stick x and the left stick y and the right stick in the middle then set
            ↪ all powers to zero.

        if the left stick x or the left stick y is greater than 0.05 (down and right on the
            ↪ gamepad) then use the cubic function to scale the power

        if the left stick x or the left stick y is less than 0.05 (up and left on the gamepad
            ↪ ) then use the cubic function to scale the power
         */

            if ((leftStickY == 0) & (leftStickX == 0) & (rightStickX != 0)) {
```

```
    VelocityLeftFront = rightStickX;
    VelocityRightFront = -rightStickX;
    VelocityLeftBack = rightStickX;
    VelocityRightBack = -rightStickX;

} else if ((leftStickY == 0) & (leftStickX == 0) & (rightStickX == 0)) {

    VelocityLeftFront = 0;
    VelocityRightFront = 0;
    VelocityLeftBack = 0;
    VelocityRightBack = 0;


} else {

    double turningConstant = 0;

    double FrontBack = leftStickY;
    double strafe = leftStickX;
    double rotate = rightStickX * turningConstant;

    VelocityLeftFront = FrontBack + rotate + strafe;
    VelocityRightFront = FrontBack - rotate - strafe;
    VelocityLeftBack = FrontBack + rotate - strafe;
    VelocityRightBack = FrontBack - rotate + strafe;

    //scaling so velocity doesn't go over 1

    double max = Math.abs(VelocityLeftFront);
    if (Math.abs(VelocityRightFront) > max) {
        max = Math.abs(VelocityRightFront);
    }
    if (Math.abs(VelocityLeftBack) > max) {
        max = Math.abs(VelocityLeftBack);
    }
    if (Math.abs(VelocityRightBack) > max) {
        max = Math.abs(VelocityRightBack);
    }
    if (max > 1) {
        VelocityLeftFront /= max; // velocity = velocity/max (will return 1)
        VelocityRightFront /= max;
        VelocityLeftBack /= max;
        VelocityRightBack /= max;
    }
}

/*double r = Math.hypot(leftStickX, leftStickY);//switch left to right and vice
    ↪ versa if you want right to control direction
double robotAngle = Math.atan2(leftStickY, leftStickX) - Math.PI / 4; //
    ↪ leftStickY is neg
double rightX = rightStickX;

VelocityLeftFront = r * Math.cos(robotAngle) - rightX;
VelocityRightFront = r * Math.sin(robotAngle) + rightX;
VelocityLeftBack = r * Math.sin(robotAngle) - rightX;
VelocityRightBack = r * Math.cos(robotAngle) + rightX;*/
```

```
            leftFront.setPower(VelocityLeftFront);
            rightFront.setPower(VelocityRightFront);
            leftBack.setPower(VelocityLeftBack);
            rightBack.setPower(VelocityRightBack);

        }

        try {
            handleIntake();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //handleGrippers();

        //Swich to relic mode by holding both bumpers
        if (gamepad2.right_bumper && gamepad2.left_bumper && pad2bumpersCanbeRead) {
            RelicMode = !RelicMode;
            pad2bumpersCanbeRead = false;
        }
        if (!pad2bumpersCanbeRead && (!gamepad2.right_bumper || !gamepad2.left_bumper)){
            pad2bumpersCanbeRead = true;
        }

        if (RelicMode){
            handleRelic();
        }

        else {
            try {
                handleTray();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        //handlePusher();

//        if (gamepad2.dpad_up) {
//            pusher.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
//            pusher.setPower(0.2);
//        } else if (gamepad2.dpad_down) {
//            pusher.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
//            pusher.setPower(-0.2);
//        } else {
//            pusher.setPower(0);
//        }



        /*if (!aPressed && gamepad1.a) {
            if (servoPos == 0.1) {
                servoPos = 0.9;
                servo.setPosition(servoPos);
            }
            else if (servoPos == 0.9) {
                servoPos = 0.1;
```

```
            servo.setPosition(servoPos);
        }
        aPressed = true;
    }
    else if (aPressed && !gamepad1.a) aPressed = false;*/
}

private void useDPad() {
    if (gamepad1.dpad_up) {
        leftBack.setPower(0.2);
        rightBack.setPower(0.2);
        leftFront.setPower(0.2);
        rightFront.setPower(0.2);
        dPadWasPressed = true;
    } else if (gamepad1.dpad_down) {
        leftBack.setPower(-0.2);
        rightBack.setPower(-0.2);
        leftFront.setPower(-0.2);
        rightFront.setPower(-0.2);
        dPadWasPressed = true;
    } else if (gamepad1.dpad_left) {
        leftBack.setPower(0.3);
        rightBack.setPower(-0.3);
        leftFront.setPower(-0.3);
        rightFront.setPower(0.3);
        dPadWasPressed = true;
    } else if (gamepad1.dpad_right) {
        leftBack.setPower(-0.3);
        rightBack.setPower(0.3);
        leftFront.setPower(0.3);
        rightFront.setPower(-0.3);
        dPadWasPressed = true;
    } else if (dPadWasPressed && !gamepad1.dpad_up && !gamepad1.dpad_down && !gamepad1.
        ↪ dpad_left && !gamepad1.dpad_right) {
        leftBack.setPower(0);
        rightBack.setPower(0);
        leftFront.setPower(0);
        rightFront.setPower(0);
        dPadWasPressed = false;
    }
}


//Intake Cases

private static final int INTAKE_OFF = 0, INTAKE = 1, OUTTAKE = 2;

private void handleIntake() throws InterruptedException {
    if (gamepad1.left_bumper && !intakeChanged) {
        /* if the intake is off, outtake
         * if the intake is intaking, turn it off
         * if the intake is outtaking, turn it off
         */
        switch (intakeStatus) {
            case INTAKE_OFF:
                outtake();
                break;
```

```
                case INTAKE:
                case OUTTAKE:
                    intakeOff();
                    break;
            }
            intakeChanged = true;
        }
        if (gamepad1.right_bumper && !intakeChanged) {
            /* if the intake is off, intake
             * if the intake is intaking, do nothing
             * if the intake is outtaking, intake
             */
            switch (intakeStatus) {
                case INTAKE_OFF:
                    intake();
                    break;
                case INTAKE:
                case OUTTAKE:
                    intakeOff();
                    break;
                //case OUTTAKE: break;
            }
            intakeChanged = true;
        }
        // wait until the user releases all intake-related buttons before allowing the user
            ↪ to change the intake again
        else if (!gamepad1.right_bumper && !gamepad1.left_bumper) {
            intakeChanged = false;
            /*if (intakeStatus == OUTTAKE) {
                intakeOff();
            }*/
        }

        IntakeControl.ManageGlyphCounterDataTeleOp();

        //glyphCountStates
        if (getGlyphs.glyphCount >= 2) {
            switch (glyphCountStatus) {
                case HASTWO:
                    initialTime = System.nanoTime();
                    glyphCountStatus = INTAKEOFF;
                    break;
                case INTAKEOFF:
                    elapsedTime = System.nanoTime() - initialTime;
                    if (elapsedTime > 800000000L) {
                        intakeOff();
                        getGlyphs.glyphCount = 0;
                        //glyphCountStatus = SERVOUP;
                    }
                    break;
//                case SERVOUP:
//                    elapsedTime = System.nanoTime() - initialTime;
//                    if (elapsedTime > 700000000L) {
//                        tableBackstopServo.setPosition(tableBackstopServoUp);
//                        getGlyphs.glyphCount = 0;
//                    }
//                    break;
```

```
            }
        }
        else {
            glyphCountStatus = HASTWO;
        }



    }

    // Intake methods. The three following methods standardize intaking, outtaking, and
        ↪ neither
    private void intake() {
        if (Math.abs(trayLift.getCurrentPosition()) < 500) {
//            gripper1.setPosition(0.95);
//            gripper2.setPosition(0.15);
//            grippersClosed = false;
            this.intake1.setPower(-INTAKE_POWER);
            this.intake2.setPower(INTAKE_POWER);
            intakeStatus = INTAKE;
        }
    }

    private void outtake() {
        if (Math.abs(trayLift.getCurrentPosition()) < 500) {
//            gripper1.setPosition(0.95);
//            gripper2.setPosition(0.15);
//            grippersClosed = false;
        }
        this.intake1.setPower(INTAKE_POWER);
        this.intake2.setPower(-INTAKE_POWER);
        intakeStatus = OUTTAKE;
    }

    private void intakeOff() {
        this.intake1.setPower(0);
        this.intake2.setPower(0);
        intakeStatus = INTAKE_OFF;
    }

    private void handleRelic() {
        /*TODO: may need to move relic servo some based on how far the arm is extended
        it tends to be a touch high if the arm is not extended fully*/
        telemetry.addLine("RELIC_MODE");
        telemetry.update();


        G2DpadUPIsPressed = gamepad2.dpad_up;

        if (G2DpadUPIsPressed) {
            G2DpadUPIsPressed = true;
            G2DpadDOWNIsPressed = false;
        } else {
            G2DpadDOWNIsPressed = gamepad2.dpad_down;
            if (G2DpadDOWNIsPressed) {
                G2DpadDOWNIsPressed = true;
                G2DpadUPIsPressed = false;
```

```
    }
}

//making sure each button press is read once
if (!G2DpadUPIsPressed) {
    G2DpadUPCanBePressed = true;
}
if (!G2DpadDOWNIsPressed) {
    G2DpadDOWNCanBePressed = true;
}


if (G2DpadUPIsPressed && G2DpadUPCanBePressed) {
    currentRelicArmPos = currentRelicArmPos + 0.02;
    relicArmServo.setPosition(currentRelicArmPos);
    telemetry.update();
    G2DpadUPCanBePressed = false;
} else if (G2DpadDOWNIsPressed && G2DpadDOWNCanBePressed) {
    currentRelicArmPos = currentRelicArmPos - 0.02;
    relicArmServo.setPosition(currentRelicArmPos);
    telemetry.update();
    G2DpadDOWNCanBePressed = false;
}


int relicPos = relic.getCurrentPosition();

relicCanExtend = true;
relicCanRetract = true;

if (relicPos < 35 || relicPos > 2230) { //this is for the new slide thing (old one is
    ↪  4445)
    if (relicPos < 35) {
        relicCanRetract = false;
        if (!ExtendingRelic) {
            relic.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
            relic.setPower(0);
        }
    }
    if (relicPos > 2230) {
        relicCanExtend = false;
        if (ExtendingRelic) {
            relic.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
            relic.setPower(0);
        }
    }

} else {
    relic.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
}


G2leftStickY = -gamepad2.left_stick_y;

if ((G2leftStickY >= -GamepadDeadZone) & (G2leftStickY <= GamepadDeadZone)) { //if
    ↪ stick is in dead zone set value equal to zero
    G2leftStickY = 0;
```

```
} else if ((G2leftStickY >= GamepadDeadZone) & (G2leftStickY <= 1)) { //if stick is
    ↪ pos use scaling with pos y-int
    //                leftStickY = ScalingFunctionPositive(leftStickY);
    G2leftStickY = ScalingFunctionPositiveCubic(G2leftStickY);
} else if ((G2leftStickY <= -GamepadDeadZone) & (G2leftStickY >= -1)) { //if stick is
    ↪  neg us scaling with neg y-int
    //leftStickY = ScalingFunctionNegative(leftStickY);
    G2leftStickY = ScalingFunctionNegativeCubic(G2leftStickY);
}

if (Math.abs(gamepad2.left_stick_y) > GamepadDeadZone) {
    relic.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
    if (G2leftStickY > 0 && !relicCanExtend){
        G2leftStickY = 0;
    }
    else if (G2leftStickY <= 0 && !relicCanRetract){
        G2leftStickY = 0;
    }

    if (G2leftStickY > 0){
        ExtendingRelic = true;
    }
    else {
        ExtendingRelic = false;
    }
    leftStickActive = true;
}
else {
    G2leftStickY = 0;
    leftStickActive = false;
}

if (gamepad2.left_stick_y > 0){
    G2leftStickY = G2leftStickY * 0.6;
}

relic.setPower(G2leftStickY);

//Relic buttons
aButtonIsPressed = gamepad2.a; //green button

if (aButtonIsPressed) {
    aButtonIsPressed = true;
    bButtonIsPressed = false;
    yButtonIsPressed = false;
    xButtonIsPressed = false;
} else {
    bButtonIsPressed = gamepad2.b; //red button
    if (bButtonIsPressed) {
        bButtonIsPressed = true;
        yButtonIsPressed = false;
        aButtonIsPressed = false;
        xButtonIsPressed = false;
    } else {
        yButtonIsPressed = gamepad2.y; //yellow button
        if (yButtonIsPressed) {
            yButtonIsPressed = true;
```

```
            bButtonIsPressed = false;
            aButtonIsPressed = false;
            xButtonIsPressed = false;
        } else {
            xButtonIsPressed = gamepad2.x; //blue button
            if (xButtonIsPressed) {
                yButtonIsPressed = false;
                bButtonIsPressed = false;
                aButtonIsPressed = false;
                xButtonIsPressed = true;
            }
        }
    }
}

//making sure each button press is read once
if (!aButtonIsPressed) {
    aButtonCanBePressed = true;
}

if (!bButtonIsPressed) {
    bButtonCanBePressed = true;
}

if (!yButtonIsPressed) {
    yButtonCanBePressed = true;
}

if (!xButtonIsPressed) {
    xButtonCanBePressed = true;
}


if (aButtonIsPressed && aButtonCanBePressed){
    currentRelicArmPos = RelicArmDown;
    relicArmServo.setPosition(currentRelicArmPos);
    aButtonCanBePressed = false;
}
if (bButtonIsPressed && bButtonCanBePressed){
    currentRelicArmPos = RelicArmNearDown;
    relicArmServo.setPosition(currentRelicArmPos);
    bButtonCanBePressed = false;
}
if (yButtonIsPressed && yButtonCanBePressed){
    currentRelicArmPos = RelicArmUp;
    relicArmServo.setPosition(currentRelicArmPos);
    yButtonCanBePressed = false;
}
if (xButtonIsPressed && xButtonCanBePressed){
    currentRelicArmPos = RelicArmNearDown;
    relicArmServo.setPosition(RelicArmNearDown);
    try {
        Thread.sleep(800);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    relicArmServo.setPosition(RelicArmNearDown + 0.06);
```

```
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            relicArmServo.setPosition(RelicArmNearDown + 0.02);
            xButtonCanBePressed = false;
        }
        if (gamepad2.right_trigger > 0.1){
            relicGrabberServo.setPosition(GrabberClosed);
        }
        if (gamepad2.left_trigger > 0.1){
            relicGrabberServo.setPosition(GrabberOpen);
        }
//        extend slides fully - run motor to pos
//                    set arm servo to down pos
//                    release relic
//                            pull up arm
//                    retract slides
    }

    //use to move relic arm up or down
//    private void useRelicDpad() {
//        telemetry.update();
//        if (gamepad2.dpad_up) {
//            currentRelicArmPos = currentRelicArmPos + 0.05;
//            telemetry.update();
//        } else if (gamepad1.dpad_down) {
//            currentRelicArmPos = currentRelicArmPos - 0.05;
//            telemetry.update();
//        }
//        relicArmServo.setPosition(currentRelicArmPos);
//    }

    private void handleTray() throws InterruptedException {

        int trayVerticalPos = trayLift.getCurrentPosition();
        //double trayTiltPos = rightTilt.getPosition();

        // reset variables
        trayCanGoUp = true;
        trayCanGoDown = true;
        trayCanTiltUp = true;
        trayCanTiltDown = true;//traySensor = false;
//
//        if (trayTouch.getState() == false) {
//            traySensor = true;
//        } else {
//            traySensor = false;
//        }

        if (trayVerticalPos < -600 || (rightTilt.getPosition() > RightServoDegreesToServoPos
            ↪ (30))) {
            if (canTurnIntakeOff) {
                intake1.setPower(0);
                intake2.setPower(0);
                intakeStatus = INTAKE_OFF;
```

```
            }
            canTurnIntakeOff = false;
        } else {
            canTurnIntakeOff = true;
        }

        //Absolute max for the table to go up so we don't break things
        if (trayVerticalPos > -5 || trayVerticalPos < -1305) { // || traySensor use later
            if (trayVerticalPos > -5) {
                trayCanGoDown = false;
                if (!raisingTray) {
                    trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
                    trayLift.setPower(0);
                }
            }
            if (trayVerticalPos < -1305) {
                trayCanGoUp = false;
                if (raisingTray) {
                    trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
                    trayLift.setPower(0);
                }
            }
            //else if (traySensor) trayCanGoDown = false;

        } else {
            trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
        }


        //manual raise
        if (Math.abs(gamepad2.right_stick_y) > 0.1) {
            double rightstick = -gamepad2.right_stick_y;
            if (rightstick > 0 && !trayCanGoDown) {
                rightstick = 0;
            } else if (rightstick <= 0 && !trayCanGoUp) {
                rightstick = 0;
            }
//              trayTilt.setMode(DcMotor.RunMode.RUN_TO_POSITION);
//              trayTilt.setTargetPosition(horizontalPos);
//              trayTilt.setPower(0.3);
            // trayTilt.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE);
            trayLift.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
            trayLift.setPower(rightstick / 2); // divided by 2 temp for testing
            trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.BRAKE); //sets motor to
                ↪ brake so table doesn't slide
            presets = false;
            rightStickActive = true;
        } else {
            rightStickActive = false;
        }
        //manual tilt
        if (Math.abs(gamepad2.left_stick_y) > 0.1) {
            double leftstick = -gamepad2.left_stick_y;
            if (leftstick > 0 && !trayCanTiltDown) leftstick = 0;
            else if (leftstick <= 0 && !trayCanTiltUp) leftstick = 0;
            rightTilt.setPosition(leftstick);
            presets = false;
```

```
    leftStickActive = true;
} else {
    leftStickActive = false;
}

//Table Presets

aButtonIsPressed = gamepad2.a; //green button

if (aButtonIsPressed) {
    aButtonIsPressed = true;
    bButtonIsPressed = false;
    yButtonIsPressed = false;
} else {
    bButtonIsPressed = gamepad2.b; //red button
    if (bButtonIsPressed) {
        bButtonIsPressed = true;
        yButtonIsPressed = false;
        aButtonIsPressed = false;
    } else {
        yButtonIsPressed = gamepad2.y; //yellow button
        if (yButtonIsPressed) {
            yButtonIsPressed = true;
            bButtonIsPressed = false;
            aButtonIsPressed = false;

        }
    }
}

//making sure each button press is read once
if (!aButtonIsPressed) {
    aButtonCanBePressed = true;
}

if (!bButtonIsPressed) {
    bButtonCanBePressed = true;
}

if (!yButtonIsPressed) {
    yButtonCanBePressed = true;
}

//bottom preset
if (aButtonIsPressed && aButtonCanBePressed) { //green button
    if (!leftStickActive && !rightStickActive) {
        tableBackstopServo.setPosition(tableBackstopServoDown);
        trayLift.setMode(DcMotor.RunMode.RUN_WITHOUT_ENCODER);
        trayLift.setZeroPowerBehavior(DcMotor.ZeroPowerBehavior.FLOAT);
        lastbuttontime = System.nanoTime();
        leftTilt.setPosition(LeftServoDegreesToServoPos(-16));
        rightTilt.setPosition(RightServoDegreesToServoPos(-16));
        tilitingTray = false;
        if (trayLift.getCurrentPosition() < -40) {
            trayLift.setPower(0.55);
            Thread.sleep(200);
            trayLift.setPower(0);
```

```
                    raisingTray = true;
                }
                presets = true;
            }
            aButtonCanBePressed = false;
        }

        //top preset
        if (bButtonIsPressed && bButtonCanBePressed) { //red button
            if (!rightStickActive && (trayLift.getCurrentPosition() > -1510)) {
                lastbuttontime = System.nanoTime();
                tableBackstopServo.setPosition(tableBackstopServoUp);
                leftTilt.setPosition(LeftServoDegreesToServoPos(-16));
                rightTilt.setPosition(RightServoDegreesToServoPos(-16));
                trayLift.setMode(DcMotor.RunMode.RUN_TO_POSITION);
                trayLift.setTargetPosition(liftUp);
                trayLift.setPower(1);
                raisingTray = true;
                presets = true;
            }
            bButtonCanBePressed = false;
        }

        //tilt preset
        if (yButtonIsPressed && yButtonCanBePressed) { //yellow button
            if (!leftStickActive) {
                lastbuttontime = System.nanoTime();
                tableBackstopServo.setPosition(tableBackstopServoDown);
                leftTilt.setPosition(LeftServoDegreesToServoPos(92));
                rightTilt.setPosition(RightServoDegreesToServoPos(92));
                getGlyphs.glyphCount = 0;
                tilitingTray = true;
                presets = true;
            }
            yButtonCanBePressed = false;
        }


        if (presets && (System.nanoTime() - lastbuttontime) > 3000000000L) {
            //trayLift.setPower(0);
        }

        if (!presets && !rightStickActive) {
            trayLift.setPower(0);
        }
        if (!presets && !leftStickActive) {
            //trayTilt.setPower(0);
        }
    }
}
//    // gripper methods
//    private void handleGrippers(){
//        if(gamepad2.left_bumper){
//            gripper1.setPosition(0.95);
//            gripper2.setPosition(0.15);
//            grippersClosed = false;
//        }
```

```
//        else if(gamepad2.right_bumper){
//            gripper1.setPosition(0.75);
//            gripper2.setPosition(0.35);
//            grippersClosed = true;
//        }
//    }



/*
gamepad1:
    sticks: drive
    bumpers: intake
gamepad2:

    stick1: table manual
    stick2: relic
    a table down
    b table top
    y table tilt
    bay: table presets
 */
```

Near Red Autonomous (Near Blue Autonomous is very similar):

```
package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.vuforia.CameraDevice;

import org.firstinspires.ftc.robotcore.external.navigation.RelicRecoveryVuMark;
import org.firstinspires.ftc.teamcode.Move;
import org.firstinspires.ftc.teamcode.Util;

/**
 * Created by elliot on 1/3/18.
 */


@Autonomous(name = "Near_RED_Multi_-_NoVision", group = "A_JOE_auto")
//@Disabled
public class NearRedAutoMultiNoVision extends LinearOpMode {

    @Override
    public void runOpMode() throws InterruptedException {

        Util.init(this); //imu initialized in here
        IntakeControl.init();
        KnockOffJewel.init(this);
        Vuforia.init(this);
        RelicRecoveryVuMark vuMark = null;
        Util.resetEncoders();

        VuforiaGoToColumn.columnState vuforiaColumn;
        VuforiaGoToColumn.columnState goToThisColumn;


      /*
      I don't Reset the encoders anywhere - doesn't seem to be a problem yet
      Don't have anywhere that sets motors to brake
       */

        telemetry.addLine("READY_FOR_START");
        telemetry.update();

        waitForStart();

        int i = 0;
        while (i < 60){
            Vuforia.VuMarkID(this);
            VuforiaGoToColumn.vuMark = Vuforia.VuMarkID(this);
            i+=1;
            Thread.sleep(10);
        }
        CameraDevice.getInstance().setFlashTorchMode(false) ;


        KnockOffJewel.KnockOffBlueJewel(this);
        DriveOffStone.NearRed(this);
        Move.moveUntilRedPIDDualProcesses(90, 0.3, this);
```

```
        /**
         * IF I CHANGE ANYTHING HERE CHANGE IN OTHER CORRESPONDING AUTOS ALSO
         */

        vuforiaColumn = VuforiaGoToColumn.NearRed();

        Move.strafeAngle(0, 0.35, 550, false); //change brake to false after test

        getGlyphs.RunWithStatesRED();

        Thread.sleep(500);

        goToThisColumn = goToColumnBasedOnMultiGlyphs.NearRed(vuforiaColumn, this);

        if (goToThisColumn != VuforiaGoToColumn.columnState.UNKNOWN){
            Move.moveSquareUpUntilRedPID(this, goToThisColumn);
        }
        else {
            Move.moveUntilRedPID(180, 0.3, this);
            Move.strafeAngle(0, 0.2, 50, true);
            Thread.sleep( 20000);
        }

        DistanceSensorLineUp.lineUpRed(true);



        PutGlyphInBox.putInBoxWithPush(Util.liftTrayforMulti);

        Util.distanceSensorArm.setPosition(Util.distanceSensorArmUp);

        Thread.sleep(20000);

        //Move.strafeAngle(270, 0.3, 30, true);
//            centers = cryptoDetect.forceFindCenters(this);
//            telemetry.update();
        //cryptoDetect.centerAlign(this,centers);


    }
}
```

Far Red Autonomous (Far Blue Autonomous is very similar):

```
package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.Autonomous;
import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.vuforia.CameraDevice;

import org.firstinspires.ftc.robotcore.external.navigation.RelicRecoveryVuMark;
import org.firstinspires.ftc.teamcode.Move;
import org.firstinspires.ftc.teamcode.Util;

/**
 * Created by lulzbot on 4/10/18.
 */

@Autonomous(name = "FarRedMultiGlyph", group = "Far_Auto")
public class FarRedMultiGlyph extends LinearOpMode {
    @Override
    public void runOpMode() throws InterruptedException {

        Util.init(this); //imu initialized in here
        IntakeControl.init();
        KnockOffJewel.init(this);
        Vuforia.init(this);
        RelicRecoveryVuMark vuMark = null;
        Util.resetEncoders();

        VuforiaGoToColumn.columnState vuforiaColumn;
        VuforiaGoToColumn.columnState columnTarget;

  /*
  I don't Reset the encoders anywhere - doesn't seem to be a problem yet
  Don't have anywhere that sets motors to brake
   */

        telemetry.addLine("READY_FOR_START");
        telemetry.update();

        waitForStart();

        int i = 0;
        while (i < 60){
            Vuforia.VuMarkID(this);
            VuforiaGoToColumn.vuMark = Vuforia.VuMarkID(this);
            i+=1;
            Thread.sleep(10);
        }
        CameraDevice.getInstance().setFlashTorchMode(false) ;

        KnockOffJewel.KnockOffBlueJewel(this);
        DriveOffStone.FarRed(this);
        //Move.strafeAngle(0,0.3, 100, false);
        Move.moveUntilRedPIDDualProcesses(0, 0.25, this);

        Move.rotateCounterClockwise(88); //rotate to face cryptobox
        //Move.strafeAngleWithoutPID(180,0.2, 50);
```

```
        vuforiaColumn = VuforiaGoToColumn.FarRed();

        //will not? need
        //KnockOffJewel.servo.setPosition(0.19);
        getGlyphs.RunWithStatesFarRed();
        columnTarget = goToColumnBasedOnMultiGlyphs.NearRed(vuforiaColumn, this);
        Util.distanceSensorArm.setPosition(Util.distanceSensorArmThreeQuartersDown);
        Move.strafeAngleWithHeading(180, 0.3, 150, false, -88);
        Move.moveUntilRedPIDwithHeading(180, 0.2, this, -88);
        if(columnTarget != VuforiaGoToColumn.columnState.UNKNOWN){
            if(columnTarget == VuforiaGoToColumn.columnState.RIGHT){
                Move.strafeAngleWithHeading(-90, 0.4, 675, true, -88);
            }
            if(columnTarget == VuforiaGoToColumn.columnState.CENTER){
                Move.strafeAngleWithHeading(-90, 0.4, 380, true, -88 );
            }
            //Move.strafeAngleWithHeading(0, 0.3, 75, false, -88);
            DistanceSensorLineUp.lineUpWithHeading(true, -88);
            PutGlyphInBox.putInBoxWithPush();
        }
        else{
            Move.strafeAngleWithHeading(0, 0.2, 50, true, -88);
        }

        Util.setAllPowers(0);

    }

}
```

Move class:

```
package org.firstinspires.ftc.teamcode;

import android.graphics.Color;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;
import com.qualcomm.robotcore.hardware.DcMotor;
import com.qualcomm.robotcore.util.Range;

import org.firstinspires.ftc.robotcore.external.navigation.AngleUnit;
import org.firstinspires.ftc.robotcore.external.navigation.AxesOrder;
import org.firstinspires.ftc.robotcore.external.navigation.AxesReference;
import org.firstinspires.ftc.robotcore.external.navigation.Orientation;
import org.firstinspires.ftc.teamcode.RelicRecovery.ColorSensor;
import org.firstinspires.ftc.teamcode.RelicRecovery.IntakeControl;
import org.firstinspires.ftc.teamcode.RelicRecovery.MecanumTeleop;
import org.firstinspires.ftc.teamcode.RelicRecovery.Vuforia;
import org.firstinspires.ftc.teamcode.RelicRecovery.VuforiaGoToColumn;
import org.firstinspires.ftc.teamcode.RelicRecovery.getGlyphs;

public class Move {

    private Move() throws Exception {
        throw new Exception();
    }

    // start the robot from a stop and gradually speed it up until <targetPower> is reached
    public static void accelerateForward(double targetPower) throws InterruptedException {
        double currentPower = 0.06;
        while ((currentPower + 0.06) < targetPower) {
            Util.setAllPowers(Range.clip(currentPower, -1, 1));
            currentPower += 0.06;
            Thread.sleep(20);
        }
        Util.setAllPowers(targetPower);
    }

    public static void accelerateBackward(double targetPower) throws InterruptedException {
        targetPower = Math.abs(targetPower);
        double currentPower = -0.06;
        while ((currentPower - 0.06) > (-targetPower)) {
            Util.setAllPowers(Range.clip(currentPower, -1, 1));
            currentPower -= 0.06;
            Thread.sleep(20);
        }
        Util.setAllPowers(-targetPower);
    }

    // slow down the robot smoothly, as opposed to jerking it to a stop
    public static void decelerateForward(double currentPower) throws InterruptedException {
        currentPower -= 0.06;
        while ((currentPower - 0.06) > 0) {
            Util.setAllPowers(Range.clip(currentPower, -1, 1));
            currentPower -= 0.06;
            Thread.sleep(20);
        }
```

```
    Util.setAllPowers(0);
}

public static void decelerateBackward(double currentPower) throws InterruptedException {
    currentPower = -Math.abs(currentPower) + 0.06;
    while ((currentPower + 0.06) < 0) {
        Util.setAllPowers(Range.clip(currentPower, -1, 1));
        currentPower += 0.06;
        Thread.sleep(20);
    }

    Util.setAllPowers(0);
}

public static void accelerateRight(double targetPower) throws InterruptedException {
    targetPower = Math.abs(targetPower);
    double currentPower = 0.06;
    while ((currentPower + 0.06) < targetPower){
        Util.rightFront.setPower(-Range.clip(currentPower, -1, 1));
        Util.rightBack.setPower(Range.clip(currentPower, -1, 1));
        Util.leftFront.setPower(Range.clip(currentPower, -1, 1));
        Util.leftBack.setPower(-Range.clip(currentPower, -1, 1));
        currentPower += 0.06;
        Thread.sleep(20);
    }
}

public static void decelerateRight(double currentPower) throws InterruptedException {
    while ((currentPower + 0.06) > 0){
        Util.rightFront.setPower(-Range.clip(currentPower, -1, 1));
        Util.rightBack.setPower(Range.clip(currentPower, -1, 1));
        Util.leftFront.setPower(Range.clip(currentPower, -1, 1));
        Util.leftBack.setPower(-Range.clip(currentPower, -1, 1));
        currentPower -= 0.06;
        Thread.sleep(20);
    }

    Util.setAllPowers(0);
}

public static void accelerateLeft(double targetPower) throws InterruptedException {
    targetPower = Math.abs(targetPower);
    double currentPower = 0.06;
    while ((currentPower + 0.06) < targetPower){
        Util.rightBack.setPower(-Range.clip(currentPower, -1, 1));
        Util.rightFront.setPower(Range.clip(currentPower, -1, 1));
        Util.leftBack.setPower(Range.clip(currentPower, -1, 1));
        Util.leftFront.setPower(-Range.clip(currentPower, -1, 1));
        currentPower += 0.06;
        Thread.sleep(20);
    }
}

public static void decelerateLeft(double currentPower) throws InterruptedException {
    while ((currentPower + 0.06) < 0){
        Util.rightBack.setPower(-Range.clip(currentPower, -1, 1));
        Util.rightFront.setPower(Range.clip(currentPower, -1, 1));
```

```
        Util.leftBack.setPower(Range.clip(currentPower, -1, 1));
        Util.leftFront.setPower(-Range.clip(currentPower, -1, 1));
        currentPower -= 0.06;
        Thread.sleep(20);
    }

    Util.setAllPowers(0);
}

// accelerate up to the target power but continue a specified distance, then stop if
    ↪ needed
public static void startForward(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    accelerateForward(power);
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateForward(power);
}

public static void startBackward(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    accelerateBackward(power);
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateBackward(power);
}

public static void startRight(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    //accelerateRight(power);
    Util.rightFront.setPower(-power);
    Util.rightBack.setPower(power);
    Util.leftFront.setPower(power);
    Util.leftBack.setPower(-power);
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) {
        Thread.sleep(20);
    }

    //if (stop) decelerateRight(power);
}

public static void startLeft(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.leftFront.getCurrentPosition();
   // accelerateLeft(power);
    Util.rightFront.setPower(power);
    Util.rightBack.setPower(-power);
    Util.leftFront.setPower(-power);
    Util.leftBack.setPower(power);
    while (Math.abs((Util.leftFront.getCurrentPosition() - pos)) < dist) {
        Thread.sleep(20);
    }
   // if (stop) decelerateLeft(power);
}
```

```
public static void continueForward(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateForward(power);
}

public static void continueBackward(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateBackward(power);
}

public static void continueRight(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.rightFront.getCurrentPosition();
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateRight(power);
}

public static void continueLeft(double power, int dist, boolean stop) throws
    ↪ InterruptedException {
    int pos = Util.leftFront.getCurrentPosition();
    while (Math.abs((Util.rightFront.getCurrentPosition() - pos)) < dist) Thread.sleep
        ↪ (20);
    if (stop) decelerateLeft(power);
}

public static void rotateCounterClockwiseForMultiGlyph() throws InterruptedException {
    long initialTime;

    Util.leftFront.setPower(-0.6);
    Util.leftBack.setPower(-0.6);
    Util.rightBack.setPower(0.6);
    Util.rightFront.setPower(0.6);

    initialTime = System.nanoTime();

    while (System.nanoTime() - initialTime < 240000000L){
        Thread.sleep(10);
        IntakeControl.ManageGlyphCounterData();
        if (getGlyphs.glyphCount > 1){
            break;
        }
    }

    Util.setAllPowers(0);
}
public static void rotateClockwiseForMultiGlyph() throws InterruptedException {
    long initialTime;

    Util.leftFront.setPower(0.6);
```

```
        Util.leftBack.setPower(0.6);
        Util.rightBack.setPower(-0.6);
        Util.rightFront.setPower(-0.6);

        initialTime = System.nanoTime();

        while (System.nanoTime() - initialTime < 240000000L){
            Thread.sleep(10);
            IntakeControl.ManageGlyphCounterData();
            if (getGlyphs.glyphCount > 1){
                break;
            }
        }

        Util.setAllPowers(0);
    }
    public static void rotateCounterClockwise(double heading) throws InterruptedException {
        Orientation angles = Util.imu.getAngularOrientation(AxesReference.INTRINSIC,
            ↪ AxesOrder.ZYX, AngleUnit.DEGREES);
        float pos = angles.firstAngle;

        //float target = -pos + degrees; // this is to try and make it so that you can
            ↪ theoretically do two turns back to back

        double finalpower = 0.35; //don't change
        double minPower = 0.25; //don't change // oops i changed it, was 0.15
        double currentPower = 0.05; //don't change

        int decelerationDegrees = 40; // change this to tweak turns

        while ((currentPower < finalpower)) { // this loop takes 1/10 of a second total. To
            ↪ tweak turns that time may change
            Util.leftFront.setPower(-currentPower);
            Util.leftBack.setPower(-currentPower);
            Util.rightBack.setPower(currentPower);
            Util.rightFront.setPower(currentPower);
            currentPower += .025;

            Thread.sleep(10);
        }

        Util.leftFront.setPower(-finalpower);
        Util.leftBack.setPower(-finalpower);
        Util.rightBack.setPower(finalpower);
        Util.rightFront.setPower(finalpower);


        while ((heading - decelerationDegrees) > (Util.imu.getAngularOrientation().firstAngle
            ↪ )){
            Thread.sleep(5);
        }

        while (currentPower > minPower){ // this loop takes 1/10 of a second total. To tweak
            ↪ turns that time may change. Needs to finish ramping down before angle is
            ↪ reached
            Util.leftFront.setPower(-currentPower);
            Util.leftBack.setPower(-currentPower);
```

```
            Util.rightBack.setPower(currentPower);
            Util.rightFront.setPower(currentPower);
            currentPower -= .02;

            Thread.sleep(10);
        }

        while ((heading) > (Util.imu.getAngularOrientation().firstAngle)){ //wait till reach
            ↪ angle (remember - we are going slower now) then stop
            Thread.sleep(5);
        }

        Util.telemetry("IMU_Data", Util.imu.getAngularOrientation().firstAngle);
        Util.linearOpMode.telemetry.update();

        Util.setAllPowers(0);
    }

    public static void rotateClockwise(double heading) throws InterruptedException {
        Orientation angles = Util.imu.getAngularOrientation(AxesReference.INTRINSIC,
            ↪ AxesOrder.ZYX, AngleUnit.DEGREES);
        float pos = angles.firstAngle;

        //float target = pos + degrees; // this is to try and make it so that you can
            ↪ theoretically do two turns back to back
        // target is difference between where you are and the degrees you want to go

        double finalpower = 0.35; //don't change
        double minPower = 0.25; //don't change // oops i changed it, was 0.15
        double currentPower = 0.05; //don't change

        int decelerationDegrees = 40; // change this to tweak turns

        while ((currentPower < finalpower)) { // this loop takes 1/10 of a second total. To
            ↪ tweak turns that time may change
            Util.leftFront.setPower(currentPower);
            Util.leftBack.setPower(currentPower);
            Util.rightBack.setPower(-currentPower);
            Util.rightFront.setPower(-currentPower);
            currentPower += .025;

            Thread.sleep(10);
        }

        Util.leftFront.setPower(finalpower);
        Util.leftBack.setPower(finalpower);
        Util.rightBack.setPower(-finalpower);
        Util.rightFront.setPower(-finalpower);

        while ((heading - decelerationDegrees > -(Util.imu.getAngularOrientation().firstAngle
            ↪ ))) {
            Thread.sleep(5);
        }

        while (currentPower > minPower) { // this loop takes 1/10 of a second total. To tweak
            ↪  turns that time may change. Needs to finish ramping down before angle is
            ↪ reached
```

```
        Util.leftFront.setPower(currentPower);
        Util.leftBack.setPower(currentPower);
        Util.rightBack.setPower(-currentPower);
        Util.rightFront.setPower(-currentPower);
        currentPower -= .02;

        Thread.sleep(10);
    }

    while ((heading) > -(Util.imu.getAngularOrientation().firstAngle)) { //wait till
        ↪ reach angle (remember - we are going slower now) then stop
        Thread.sleep(5);
    }
    Util.telemetry("IMU_Data", Util.imu.getAngularOrientation().firstAngle);
    Util.linearOpMode.telemetry.update();

    Util.setAllPowers(0);
}

public static boolean strafeAngle(double angle, double power, int dist, boolean brake)
    ↪ throws InterruptedException {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    while (((Math.abs(Util.leftBack.getCurrentPosition() - lB)
            + Math.abs(Util.rightBack.getCurrentPosition() - rB)
            + Math.abs(Util.leftFront.getCurrentPosition() - lF)
            + Math.abs(Util.rightFront.getCurrentPosition() - rF)) / 4) < dist) {
    //for (int i = 0; i < 75; i++) {
        rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
```

```
        /**/

        /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        Thread.sleep(20);
    }

    if (brake) {
        Util.setDriveModeBrake();
    }
    else {
        Util.setDriveModeFloat();
    }

    Util.setAllPowers(0);

    return true;
}
public static boolean strafeAngle(double angle, double power) throws InterruptedException
    ↪  {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);


    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;



    rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
```

```
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
        /**/

    Thread.sleep(10);

    return true;
}

public static boolean strafeAngleforTime (double angle, double power, double seconds)
    ↪ throws InterruptedException {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    double i = 0;
    while (i < seconds) {
        //for (int i = 0; i < 75; i++) {
        rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        i += 0.02; // because it takes 2/100 of a second to complete loop
```

```
        Thread.sleep(20);
    }

    Util.setDriveModeFloat();
    Util.setAllPowers(0);

    return true;
}

public static boolean strafeAngleforTimeWithoutPID (double angle, double power, double
    ↪ seconds) throws InterruptedException {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    double i = 0;
    while (i < seconds) {
        //for (int i = 0; i < 75; i++) {
        rotate = 0;

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        i += 0.02; // because it takes 2/100 of a second to complete loop
```

```
        Thread.sleep(20);
    }

    Util.setDriveModeFloat();
    Util.setAllPowers(0);

    return true;
}

protected static float RightAmountofRed;
protected static float RightAmountofBlue;

protected static float LeftAmountofRed;
protected static float LeftAmountofBlue;

public static boolean moveUntilRedPID (double angle, double power, LinearOpMode opMode)
    ↪ throws InterruptedException {
    ColorSensor.init(opMode);
    ColorSensor.ReadSensor(opMode);

    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    RightAmountofRed = Color.red(ColorSensor.RightColor);
    RightAmountofBlue = Color.blue(ColorSensor.RightColor);

    LeftAmountofRed = Color.red(ColorSensor.LeftColor);
    LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

    double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
    double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

    while (rightRatio < 1.5 && leftRatio < 1.5) {
        ColorSensor.ReadSensor(opMode);
        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);
```

```
            rightRatio = RightAmountofRed/((double)RightAmountofBlue);

            LeftAmountofRed = Color.red(ColorSensor.LeftColor);
            LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
            leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

            rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

             /**/
            Util.leftFront.setPower(leftFront + rotate);
            Util.rightFront.setPower(rightFront - rotate);
            Util.leftBack.setPower(leftBack + rotate);
            Util.rightBack.setPower(rightBack - rotate);
            /**/

            Thread.sleep(20);
        }

        Util.setAllPowers(0);

        return true;
    }

    public static double moveSquareUpUntilRedPID (LinearOpMode opMode, VuforiaGoToColumn.
        ↪ columnState column) throws InterruptedException {
        ColorSensor.init(opMode);
        ColorSensor.ReadSensor(opMode);

        boolean leftHitFirst = false;
        boolean rightHitFirst = false;

        double distanceToMiddle;
        double distanceToTravel = 0;

        double angle = 180;

        double originalAngle = 0;// PID.heading(Util.imu);

        int lB = Util.leftBack.getCurrentPosition();
        int rB = Util.rightBack.getCurrentPosition();
        int lF = Util.leftFront.getCurrentPosition();
        int rF = Util.rightFront.getCurrentPosition();

        angle = -angle + 90;

        double frontBack = 0.25 * Math.sin(Math.toRadians(angle));
        double strafe = 0.25 * Math.cos(Math.toRadians(angle));

        double rotate;

        double leftFront, rightFront, leftBack, rightBack;

        leftFront = frontBack + strafe;
        rightFront = frontBack - strafe;
        leftBack = frontBack - strafe;
        rightBack = frontBack + strafe;
```

```
RightAmountofRed = Color.red(ColorSensor.RightColor);
RightAmountofBlue = Color.blue(ColorSensor.RightColor);

LeftAmountofRed = Color.red(ColorSensor.LeftColor);
LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

while (leftRatio < 1.5 && rightRatio < 1.5) {
    ColorSensor.ReadSensor(opMode);
    LeftAmountofRed = Color.red(ColorSensor.LeftColor);
    LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
    leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

    RightAmountofRed = Color.red(ColorSensor.RightColor);
    RightAmountofBlue = Color.blue(ColorSensor.RightColor);
    rightRatio = RightAmountofRed/((double)RightAmountofBlue);

    rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

     /**/
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
    /**/

    Thread.sleep(10);
}

Util.setAllPowers(0);

if (leftRatio > 1.5) { //left sensor saw it
    leftHitFirst = true;
    Util.resetEncoders();
    while (rightRatio < 1.5) {
        ColorSensor.ReadSensor(opMode);

        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);
        rightRatio = RightAmountofRed/((double)RightAmountofBlue);

        strafeAngle(-90, 0.3);

        Thread.sleep(10);
    }
    Util.setAllPowers(0);
    distanceToMiddle = 0.5 * ((Math.abs(Util.rightFront.getCurrentPosition()) + Math.
        ↪ abs(Util.rightBack.getCurrentPosition()) + Math.abs(Util.leftFront.
        ↪ getCurrentPosition()) + Math.abs(Util.leftBack.getCurrentPosition())) / 4)
        ↪ ;
    Util.telemetry("distance_to_middle", distanceToMiddle, true);
    if (column == VuforiaGoToColumn.columnState.LEFT){
        distanceToTravel = distanceToMiddle + 275;
    }
    else if (column == VuforiaGoToColumn.columnState.CENTER){
```

```
            strafeAngle(90, 0.4, (int) (distanceToMiddle), true);
        }
        else if (column == VuforiaGoToColumn.columnState.RIGHT){
            distanceToTravel = 405 - distanceToMiddle;
        }
    }
    else {
        rightHitFirst = true;
        Util.resetEncoders();
        while (leftRatio < 1.5) {
            ColorSensor.ReadSensor(opMode);

            LeftAmountofRed = Color.red(ColorSensor.LeftColor);
            LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
            leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

            strafeAngle(90, 0.3);

            Thread.sleep(10);
        }
        Util.setAllPowers(0);
        distanceToMiddle = 0.5 * ((Math.abs(Util.rightFront.getCurrentPosition()) + Math.
            ↪ abs(Util.rightBack.getCurrentPosition()) + Math.abs(Util.leftFront.
            ↪ getCurrentPosition()) + Math.abs(Util.leftBack.getCurrentPosition())) / 4)
            ↪ ;
        Util.telemetry("distance_to_middle", distanceToMiddle, true);
        if (column == VuforiaGoToColumn.columnState.RIGHT){
            distanceToTravel = distanceToMiddle + 325;
        }
        else if (column == VuforiaGoToColumn.columnState.CENTER){
            strafeAngle(-90, 0.4, (int) (distanceToMiddle), true);
        }
        else if (column == VuforiaGoToColumn.columnState.LEFT){
            distanceToTravel = 325 - distanceToMiddle;
        }
    }

    if (distanceToMiddle < 40){ //starts center
        strafeAngle(0, 0.3, 100, true);
        if (column == VuforiaGoToColumn.columnState.CENTER) {
            strafeAngle(-90, 0.4, 50, true);
        }
        else if (column == VuforiaGoToColumn.columnState.LEFT){
            strafeAngle(90, 0.4, 300, true);
        }
        else if (column == VuforiaGoToColumn.columnState.RIGHT) {
            strafeAngle(-90, 0.4, 350, true);
        }

    }
    else {
        if (rightHitFirst) {
            if (column == VuforiaGoToColumn.columnState.RIGHT){
                strafeAngle(-90, 0.4, (int) (distanceToTravel), true);
            }
            else if (column == VuforiaGoToColumn.columnState.CENTER) {
                strafeAngle(90, 0.4, 20, true);
```

```
            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                strafeAngle(90, 0.4, (int) (distanceToTravel), true);
            }
        }
        else if (leftHitFirst) {
            if (column == VuforiaGoToColumn.columnState.LEFT){
                strafeAngle(90, 0.4, (int) distanceToTravel, true);
            }
            else if (column == VuforiaGoToColumn.columnState.CENTER) {
                strafeAngle(-90, 0.4, 50, true);
            }
            else if (column == VuforiaGoToColumn.columnState.RIGHT){
                strafeAngle(-90, 0.4, (int) (distanceToTravel), true);
            }

        }
    }

    return distanceToMiddle;
}

public static boolean moveUntilRedPIDDualProcesses (double angle, double power,
    ↪ LinearOpMode opMode) throws InterruptedException {
    ColorSensor.init(opMode);
    ColorSensor.ReadSensor(opMode);

    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    Util.distanceSensorArm.setPosition(0.7);
    Util.leftTiltServo.setPosition(MecanumTeleop.LeftServoDegreesToServoPos(-2));
    Util.rightTiltServo.setPosition(MecanumTeleop.RightServoDegreesToServoPos(-2));
    Util.lift.setMode(DcMotor.RunMode.RUN_TO_POSITION);
    Util.lift.setTargetPosition(-762);
    Util.lift.setPower(0.5); //used to unfold the intake - needs this much power

    //put tray back down with gravity same as in teleop


    leftFront = frontBack + strafe;
```

```
        rightFront = frontBack - strafe;
        leftBack = frontBack - strafe;
        rightBack = frontBack + strafe;

        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);

        LeftAmountofRed = Color.red(ColorSensor.LeftColor);
        LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

        double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
        double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

        while (rightRatio < 1.5 && leftRatio < 1.5) {
            ColorSensor.ReadSensor(opMode);
            RightAmountofRed = Color.red(ColorSensor.RightColor);
            RightAmountofBlue = Color.blue(ColorSensor.RightColor);
            rightRatio = RightAmountofRed/((double)RightAmountofBlue);

            LeftAmountofRed = Color.red(ColorSensor.LeftColor);
            LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
            leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

            rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

             /**/
            Util.leftFront.setPower(leftFront + rotate);
            Util.rightFront.setPower(rightFront - rotate);
            Util.leftBack.setPower(leftBack + rotate);
            Util.rightBack.setPower(rightBack - rotate);
            /**/


        }

        Util.lift.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        Util.lift.setTargetPosition(-20);
        Util.lift.setPower(0.5);

        return true;
    }

    public static boolean moveUntilBluePID (double angle, double power, LinearOpMode opMode)
        ↪ throws InterruptedException {
        ColorSensor.init(opMode);
        ColorSensor.ReadSensor(opMode);

        if (power <= 0) return false;
        if (power > 0.7) power = 0.7;

        double originalAngle = 0;// PID.heading(Util.imu);

        int lB = Util.leftBack.getCurrentPosition();
        int rB = Util.rightBack.getCurrentPosition();
        int lF = Util.leftFront.getCurrentPosition();
        int rF = Util.rightFront.getCurrentPosition();
```

```
        angle = -angle + 90;

        double frontBack = power * Math.sin(Math.toRadians(angle));
        double strafe = power * Math.cos(Math.toRadians(angle));

        double rotate;

        double leftFront, rightFront, leftBack, rightBack;

        leftFront = frontBack + strafe;
        rightFront = frontBack - strafe;
        leftBack = frontBack - strafe;
        rightBack = frontBack + strafe;

        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);

        LeftAmountofRed = Color.red(ColorSensor.LeftColor);
        LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

        double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
        double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

        while (leftRatio > 0.8 && rightRatio > 0.8) {
            ColorSensor.ReadSensor(opMode);
            LeftAmountofRed = Color.red(ColorSensor.LeftColor);
            LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
            leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

            RightAmountofRed = Color.red(ColorSensor.RightColor);
            RightAmountofBlue = Color.blue(ColorSensor.RightColor);
            rightRatio = RightAmountofRed/((double)RightAmountofBlue);

            rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

             /**/
            Util.leftFront.setPower(leftFront + rotate);
            Util.rightFront.setPower(rightFront - rotate);
            Util.leftBack.setPower(leftBack + rotate);
            Util.rightBack.setPower(rightBack - rotate);
            /**/

            Thread.sleep(20);
        }

        Util.setAllPowers(0);

        return true;
    }

    public static double moveSquareUpUntilBluePID (LinearOpMode opMode, VuforiaGoToColumn.
        ↪ columnState column) throws InterruptedException {
        ColorSensor.init(opMode);
        ColorSensor.ReadSensor(opMode);

        boolean leftHitFirst = false;
        boolean rightHitFirst = false;
```

```
double distanceToMiddle;
double distanceToTravel = 0;

double angle = 180;

double originalAngle = 0;// PID.heading(Util.imu);

int lB = Util.leftBack.getCurrentPosition();
int rB = Util.rightBack.getCurrentPosition();
int lF = Util.leftFront.getCurrentPosition();
int rF = Util.rightFront.getCurrentPosition();

angle = -angle + 90;

double frontBack = 0.25 * Math.sin(Math.toRadians(angle));
double strafe = 0.25 * Math.cos(Math.toRadians(angle));

double rotate;

double leftFront, rightFront, leftBack, rightBack;

leftFront = frontBack + strafe;
rightFront = frontBack - strafe;
leftBack = frontBack - strafe;
rightBack = frontBack + strafe;

RightAmountofRed = Color.red(ColorSensor.RightColor);
RightAmountofBlue = Color.blue(ColorSensor.RightColor);

LeftAmountofRed = Color.red(ColorSensor.LeftColor);
LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

while (leftRatio > 0.8 && rightRatio > 0.8) {
    ColorSensor.ReadSensor(opMode);
    LeftAmountofRed = Color.red(ColorSensor.LeftColor);
    LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
    leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

    RightAmountofRed = Color.red(ColorSensor.RightColor);
    RightAmountofBlue = Color.blue(ColorSensor.RightColor);
    rightRatio = RightAmountofRed/((double)RightAmountofBlue);

    rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

     /**/
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
    /**/

    Thread.sleep(10);
}
```

```
Util.setAllPowers(0);

if (leftRatio < 0.8) { //left sensor saw it
    leftHitFirst = true;
    Util.resetEncoders();
    while (rightRatio > 0.8) {
        ColorSensor.ReadSensor(opMode);

        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);
        rightRatio = RightAmountofRed/((double)RightAmountofBlue);

        strafeAngle(-90, 0.3);

        Thread.sleep(10);
    }
    Util.setAllPowers(0);
    distanceToMiddle = 0.5 * ((Math.abs(Util.rightFront.getCurrentPosition()) + Math.
        ↪ abs(Util.rightBack.getCurrentPosition()) + Math.abs(Util.leftFront.
        ↪ getCurrentPosition()) + Math.abs(Util.leftBack.getCurrentPosition())) / 4)
        ↪ ;
    Util.telemetry("distance_to_middle", distanceToMiddle, true);
    if (column == VuforiaGoToColumn.columnState.LEFT){
        distanceToTravel = distanceToMiddle + 275;
    }
    else if (column == VuforiaGoToColumn.columnState.CENTER){
        strafeAngle(90, 0.4, (int) (distanceToMiddle), true);
    }
    else if (column == VuforiaGoToColumn.columnState.RIGHT){
        distanceToTravel = 405 - distanceToMiddle;
    }
}
else {
    rightHitFirst = true;
    Util.resetEncoders();
    while (leftRatio > 0.8) {
        ColorSensor.ReadSensor(opMode);

        LeftAmountofRed = Color.red(ColorSensor.LeftColor);
        LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
        leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

        strafeAngle(90, 0.3);

        Thread.sleep(10);
    }
    Util.setAllPowers(0);
    distanceToMiddle = 0.5 * ((Math.abs(Util.rightFront.getCurrentPosition()) + Math.
        ↪ abs(Util.rightBack.getCurrentPosition()) + Math.abs(Util.leftFront.
        ↪ getCurrentPosition()) + Math.abs(Util.leftBack.getCurrentPosition())) / 4)
        ↪ ;
    Util.telemetry("distance_to_middle", distanceToMiddle, true);
    if (column == VuforiaGoToColumn.columnState.RIGHT){
        distanceToTravel = distanceToMiddle + 325;
    }
    else if (column == VuforiaGoToColumn.columnState.CENTER){
```

```
                strafeAngle(-90, 0.4, (int) (distanceToMiddle), true);
            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                distanceToTravel = 325 - distanceToMiddle;
            }
        }

        if (distanceToMiddle < 40){
            strafeAngle(0, 0.3, 100, true);
            if (column == VuforiaGoToColumn.columnState.CENTER) {
                strafeAngle(-90, 0.4, 100, true);
            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                strafeAngle(90, 0.4, 300, true);
            }
            else if (column == VuforiaGoToColumn.columnState.RIGHT) {
                strafeAngle(-90, 0.4, 450, true);
            }

        }
        else {
            if (rightHitFirst) {
                if (column == VuforiaGoToColumn.columnState.RIGHT){
                    strafeAngle(-90, 0.4, (int) (distanceToTravel), true);
                }
                else if (column == VuforiaGoToColumn.columnState.CENTER) {
                    strafeAngle(90, 0.4, 20, true);
                }
                else if (column == VuforiaGoToColumn.columnState.LEFT){
                    strafeAngle(90, 0.4, (int) (distanceToTravel), true);
                }
            }
            else if (leftHitFirst) {
                if (column == VuforiaGoToColumn.columnState.LEFT){
                    strafeAngle(90, 0.4, (int) distanceToTravel, true);
                }
                else if (column == VuforiaGoToColumn.columnState.CENTER) {
                    strafeAngle(-90, 0.4, 120, true);
                }
                else if (column == VuforiaGoToColumn.columnState.RIGHT){
                    strafeAngle(-90, 0.4, (int) (distanceToTravel), true);
                }

            }
        }

        return distanceToMiddle;
    }

    public static boolean moveUntilBluePIDDualProcesses (double angle, double power,
        ↪ LinearOpMode opMode) throws InterruptedException {
        ColorSensor.init(opMode);
        ColorSensor.ReadSensor(opMode);

        if (power <= 0) return false;
        if (power > 0.7) power = 0.7;
```

```
double originalAngle = 0;// PID.heading(Util.imu);

int lB = Util.leftBack.getCurrentPosition();
int rB = Util.rightBack.getCurrentPosition();
int lF = Util.leftFront.getCurrentPosition();
int rF = Util.rightFront.getCurrentPosition();

angle = -angle + 90;

double frontBack = power * Math.sin(Math.toRadians(angle));
double strafe = power * Math.cos(Math.toRadians(angle));

double rotate;

double leftFront, rightFront, leftBack, rightBack;

Util.distanceSensorArm.setPosition(0.7);
Util.leftTiltServo.setPosition(MecanumTeleop.LeftServoDegreesToServoPos(-2));
Util.rightTiltServo.setPosition(MecanumTeleop.RightServoDegreesToServoPos(-2));
Util.lift.setMode(DcMotor.RunMode.RUN_TO_POSITION);
Util.lift.setTargetPosition(-762);
Util.lift.setPower(0.5); //used to unfold the intake - needs this much power

//put tray back down with gravity same as in teleop

leftFront = frontBack + strafe;
rightFront = frontBack - strafe;
leftBack = frontBack - strafe;
rightBack = frontBack + strafe;

RightAmountofRed = Color.red(ColorSensor.RightColor);
RightAmountofBlue = Color.blue(ColorSensor.RightColor);

LeftAmountofRed = Color.red(ColorSensor.LeftColor);
LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

while (leftRatio > 0.8 && rightRatio > 0.8) {
    ColorSensor.ReadSensor(opMode);
    LeftAmountofRed = Color.red(ColorSensor.LeftColor);
    LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
    leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

    RightAmountofRed = Color.red(ColorSensor.RightColor);
    RightAmountofBlue = Color.blue(ColorSensor.RightColor);
    rightRatio = RightAmountofRed/((double)RightAmountofBlue);

    rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);

     /**/
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
    /**/
```

```
            Thread.sleep(20);
        }

        Util.lift.setMode(DcMotor.RunMode.RUN_TO_POSITION);
        Util.lift.setTargetPosition(-20);
        Util.lift.setPower(0.5); //used to unfold the intake - needs this much power

        return true;
    }
    public static boolean strafeAngleWithoutPID (double angle, double power, int dist) throws
        ↪  InterruptedException {
        if (power <= 0) return false;
        if (power > 0.7) power = 0.7;

        double originalAngle = 0;// PID.heading(Util.imu);

        int lB = Util.leftBack.getCurrentPosition();
        int rB = Util.rightBack.getCurrentPosition();
        int lF = Util.leftFront.getCurrentPosition();
        int rF = Util.rightFront.getCurrentPosition();

        angle = -angle + 90;

        double frontBack = power * Math.sin(Math.toRadians(angle));
        double strafe = power * Math.cos(Math.toRadians(angle));

        double rotate;

        double leftFront, rightFront, leftBack, rightBack;

        leftFront = frontBack + strafe;
        rightFront = frontBack - strafe;
        leftBack = frontBack - strafe;
        rightBack = frontBack + strafe;

        while (((Math.abs(Util.leftBack.getCurrentPosition() - lB)
                + Math.abs(Util.rightBack.getCurrentPosition() - rB)
                + Math.abs(Util.leftFront.getCurrentPosition() - lF)
                + Math.abs(Util.rightFront.getCurrentPosition() - rF)) / 4) < dist) {
            //for (int i = 0; i < 75; i++) {
            //rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);
            rotate = 0;
            /*/
            Util.leftFront.setPower(leftFront);
            Util.rightFront.setPower(rightFront);
            Util.leftBack.setPower(leftBack);
            Util.rightBack.setPower(rightBack);
            /**/

            /**/
            Util.leftFront.setPower(leftFront + rotate);
            Util.rightFront.setPower(rightFront - rotate);
            Util.leftBack.setPower(leftBack + rotate);
            Util.rightBack.setPower(rightBack - rotate);
            /**/
```

```
        Thread.sleep(20);
    }

    Util.setAllPowers(0);

    return true;
}

public static boolean strafeAngleWithoutPID(double angle, double power) throws
    ↪ InterruptedException {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    double originalAngle = 0;// PID.heading(Util.imu);


    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;



    //rotate = -0.01 * (PID.heading(Util.imu) - originalAngle);
    rotate = 0;
        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
        /**/

    Thread.sleep(10);

    return true;
}
public static boolean strafeAngleWithHeading(double angle, double power, int dist,
    ↪ boolean brake, double heading) throws InterruptedException{
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;
```

```
    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    while (((Math.abs(Util.leftBack.getCurrentPosition() - lB)
            + Math.abs(Util.rightBack.getCurrentPosition() - rB)
            + Math.abs(Util.leftFront.getCurrentPosition() - lF)
            + Math.abs(Util.rightFront.getCurrentPosition() - rF)) / 4) < dist) {
        //for (int i = 0; i < 75; i++) {
        rotate = -0.01 * (PID.heading(Util.imu) - heading);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        Thread.sleep(20);
    }

    if (brake) {
        Util.setDriveModeBrake();
    }
    else {
        Util.setDriveModeFloat();
    }

    Util.setAllPowers(0);

    return true;
}
public static boolean strafeAngleWithHeading(double angle, double power, double heading)
    ↪ throws InterruptedException {
```

```
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;



    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;



    rotate = -0.01 * (PID.heading(Util.imu) - heading);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
    Util.leftFront.setPower(leftFront + rotate);
    Util.rightFront.setPower(rightFront - rotate);
    Util.leftBack.setPower(leftBack + rotate);
    Util.rightBack.setPower(rightBack - rotate);
        /**/

    Thread.sleep(10);

    return true;
}
public static boolean strafeAngleforTimeWithHeading (double angle, double power, double
    ↪ seconds, double heading) throws InterruptedException {
    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;


    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));
```

```
    double rotate;

    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    double i = 0;
    while (i < seconds) {
        //for (int i = 0; i < 75; i++) {
        rotate = -0.01 * (PID.heading(Util.imu) - heading);

        /*/
        Util.leftFront.setPower(leftFront);
        Util.rightFront.setPower(rightFront);
        Util.leftBack.setPower(leftBack);
        Util.rightBack.setPower(rightBack);
        /**/

        /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        i += 0.02; // because it takes 2/100 of a second to complete loop

        Thread.sleep(20);
    }

    Util.setDriveModeFloat();
    Util.setAllPowers(0);

    return true;
}
public static boolean moveUntilRedPIDwithHeading (double angle, double power,
    ↪ LinearOpMode opMode, double heading) throws InterruptedException {
    ColorSensor.init(opMode);
    ColorSensor.ReadSensor(opMode);

    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;

    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();

    angle = -angle + 90;

    double frontBack = power * Math.sin(Math.toRadians(angle));
    double strafe = power * Math.cos(Math.toRadians(angle));

    double rotate;
```

```
    double leftFront, rightFront, leftBack, rightBack;

    leftFront = frontBack + strafe;
    rightFront = frontBack - strafe;
    leftBack = frontBack - strafe;
    rightBack = frontBack + strafe;

    RightAmountofRed = Color.red(ColorSensor.RightColor);
    RightAmountofBlue = Color.blue(ColorSensor.RightColor);

    LeftAmountofRed = Color.red(ColorSensor.LeftColor);
    LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

    double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
    double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

    while (rightRatio < 1.5 && leftRatio < 1.5) {
        ColorSensor.ReadSensor(opMode);
        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);
        rightRatio = RightAmountofRed/((double)RightAmountofBlue);

        LeftAmountofRed = Color.red(ColorSensor.LeftColor);
        LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
        leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

        rotate = -0.01 * (PID.heading(Util.imu) - heading);
        opMode.telemetry.update();
         /**/
        Util.leftFront.setPower(leftFront + rotate);
        Util.rightFront.setPower(rightFront - rotate);
        Util.leftBack.setPower(leftBack + rotate);
        Util.rightBack.setPower(rightBack - rotate);
        /**/

        Thread.sleep(10);
    }

    Util.setAllPowers(0);

    return true;
}

public static boolean  moveUntilBluePIDWithHeading(double angle, double power,
    ↪ LinearOpMode opMode, double heading) throws InterruptedException{
    ColorSensor.init(opMode);
    ColorSensor.ReadSensor(opMode);

    if (power <= 0) return false;
    if (power > 0.7) power = 0.7;


    int lB = Util.leftBack.getCurrentPosition();
    int rB = Util.rightBack.getCurrentPosition();
    int lF = Util.leftFront.getCurrentPosition();
    int rF = Util.rightFront.getCurrentPosition();
```

```
        angle = -angle + 90;

        double frontBack = power * Math.sin(Math.toRadians(angle));
        double strafe = power * Math.cos(Math.toRadians(angle));

        double rotate;

        double leftFront, rightFront, leftBack, rightBack;

        leftFront = frontBack + strafe;
        rightFront = frontBack - strafe;
        leftBack = frontBack - strafe;
        rightBack = frontBack + strafe;

        RightAmountofRed = Color.red(ColorSensor.RightColor);
        RightAmountofBlue = Color.blue(ColorSensor.RightColor);

        LeftAmountofRed = Color.red(ColorSensor.LeftColor);
        LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);

        double rightRatio = RightAmountofRed/((double)RightAmountofBlue);
        double leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

        while (leftRatio > 0.8 && rightRatio > 0.8) {
            ColorSensor.ReadSensor(opMode);
            LeftAmountofRed = Color.red(ColorSensor.LeftColor);
            LeftAmountofBlue = Color.blue(ColorSensor.LeftColor);
            leftRatio = LeftAmountofRed/((double)LeftAmountofBlue);

            RightAmountofRed = Color.red(ColorSensor.RightColor);
            RightAmountofBlue = Color.blue(ColorSensor.RightColor);
            rightRatio = RightAmountofRed/((double)RightAmountofBlue);

            rotate = -0.01 * (PID.heading(Util.imu) - heading);

             /**/
            Util.leftFront.setPower(leftFront + rotate);
            Util.rightFront.setPower(rightFront - rotate);
            Util.leftBack.setPower(leftBack + rotate);
            Util.rightBack.setPower(rightBack - rotate);
            /**/

            Thread.sleep(20);
        }

        Util.setAllPowers(0);

        return true;
    }
}
```

Autonomous Glyphs State Machine:

```
package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;

import org.firstinspires.ftc.teamcode.Move;
import org.firstinspires.ftc.teamcode.Util;

import static org.firstinspires.ftc.teamcode.RelicRecovery.getGlyphs.glyphState.BREAK_LOOP;
import static org.firstinspires.ftc.teamcode.RelicRecovery.getGlyphs.glyphState.LEAVE_PILE;
import static org.firstinspires.ftc.teamcode.RelicRecovery.getGlyphs.glyphState.TURN_LEFT;
import static org.firstinspires.ftc.teamcode.RelicRecovery.getGlyphs.glyphState.TURN_RIGHT;

/**
 * Created by lulzbot on 2/26/18.
 */

public class getGlyphs {
    public static int glyphCount;
    private static int encoderTicks;
    private static boolean isOpen;
    private static boolean canOpen = true;
    private static boolean turnedRight;
    private static boolean turnedLeft;

    private static boolean moving;


    public enum glyphState {
        STRAIGHT, TURN_RIGHT, TURN_LEFT, LEAVE_PILE, BREAK_LOOP
    }

    public enum glyphColor {
        GREY, BROWN, UNKNOWN
    }

    private static glyphState myState;
    public static glyphColor myColor;
    public static glyphColor firstGlyphColor;
    public static glyphColor secondGlyphColor;

    private getGlyphs() throws Exception {
        throw new Exception();
    }

    static boolean onTheRight = false; //same right as column
    static boolean onTheLeft = false; //same left as column


    public static void REDNoStateMachineGetGlyphs() throws InterruptedException {
        Util.resetEncoders();
        Util.intake1.setPower(0.9);
        Util.intake2.setPower(0.9);
        Move.strafeAngle(0, 0.25, 300, true); //6in?
        Move.rotateClockwiseForMultiGlyph();
        Thread.sleep(100);
        Move.strafeAngleWithoutPID(0, 0.25, 200); //2in?
```

```
        Thread.sleep(500);
        Util.intake1.setPower(-0.9);
        Util.intake2.setPower(-0.9);
        Thread.sleep(250);
        Util.intake1.setPower(0);
        Util.intake2.setPower(0);
        Move.strafeAngleWithoutPID(180, 0.25, 200); //2in?
        Move.rotateCounterClockwiseForMultiGlyph();
        Thread.sleep(100);
        Move.strafeAngle(180, 0.25, 300, true);
        Move.strafeAngle(90, 0.3, 425, true); //should put robot in center
    }
    public static void BLUENoStateMachineGetGlyphs() throws InterruptedException {
        Util.resetEncoders();
        Util.intake1.setPower(0.9);
        Util.intake2.setPower(0.9);
        Move.strafeAngle(0, 0.25, 300, true); //6in?
        Move.rotateCounterClockwiseForMultiGlyph();
        Thread.sleep(100);
        Move.strafeAngleWithoutPID(0, 0.25, 200); //2in?
        Thread.sleep(500);
        Util.intake1.setPower(-0.9);
        Util.intake2.setPower(-0.9);
        Thread.sleep(250);
        Util.intake1.setPower(0);
        Util.intake2.setPower(0);
        Move.strafeAngleWithoutPID(180, 0.25, 200); //2in?
        Move.rotateClockwiseForMultiGlyph();
        Thread.sleep(100);
        Move.strafeAngle(180, 0.25, 300, true);
        Move.strafeAngle(-90, 0.3, 400, true); //should put robot in center
    }

    public static void RunWithStatesRED() throws InterruptedException {
        Util.resetEncoders();
        glyphCount = 0;
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        IntakeControl.ManageGlyphCounterData();
        myState = glyphState.STRAIGHT;
        turnedRight = false;
        turnedLeft = false;
        moving = false;

        while (myState != BREAK_LOOP) {
            switch (myState) {
                case STRAIGHT:
                    //Move into pile and try to collect
                    //glyphCounter(); //Get glyph count
                    if (!moving) {
                        Move.strafeAngle(0, 0.25);
                        Thread.sleep(5);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }
```

```
            IntakeControl.ManageGlyphCounterData();
           // IntakeControl.ManageDataAndHandleStalls();
            //Get glyph count

            if (glyphCount == 2) {
                //if we have 2 glyphs, leave
                Util.intake1.setPower(0);
                Util.intake2.setPower(0);

                //added without testing
                IntakeControl.ManageGlyphCounterData();

                //myState = LEAVE_PILE;
            }

            if (Util.rightFront.getCurrentPosition() > 400) { // cut out to improve
                ↪ proformance glyphCount == 1 ||
                //if we have gone over our encoder position go to the next collection
                myState = TURN_RIGHT;
                Move.rotateClockwiseForMultiGlyph();
                Util.resetEncoders();
                moving = false;
            }
            break;
        case TURN_RIGHT:
            //try to collect after turning right
            //glyphCounter(); //Get glyph count
            turnedRight = true; //we need to turn left leaving
            if (!moving) {
                Move.strafeAngleWithoutPID(0, 0.25);
                Thread.sleep(5);
                moving = true;
            } else {
                Thread.sleep(10);
            }

            IntakeControl.ManageGlyphCounterData();
            //IntakeControl.ManageDataAndHandleStalls();

            if (glyphCount == 2) {
                //if we have 2 glyphs
                //myState = LEAVE_PILE;
                Util.intake1.setPower(0);
                Util.intake2.setPower(0);

                //added without testing
                IntakeControl.ManageGlyphCounterData();

            }

            if (Util.rightFront.getCurrentPosition() > 400) {
                myState = LEAVE_PILE;
               // Move.rotateCounterClockwiseForMultiGlyph();
                //use this rotation if consistently only get one
                Util.telemetry("glyphs", glyphCount, true);
                Util.resetEncoders();
                moving = false;
```

```
                        // (hopefully by this point we have 2)
                    }
                    break;
//                case TURN_LEFT:
//                    //glyphCounter(); //Get glyph count
//                    if (VuforiaGoToColumn.column != VuforiaGoToColumn.columnState.LEFT){
//                        Util.telemetry("Turn Left!!", 0);
//                        turnedLeft = true; //used to help with leaving pile
//                        //try to collect after turning left
//                        if (!moving) {
//                            Move.strafeAngleWithoutPID(0, 0.25);
//                            moving = true;
//                        } else {
//                            Thread.sleep(10);
//                        }
//                        IntakeControl.ManageGlyphCounterData();
//                        IntakeControl.ManageDataAndHandleStalls();
//
//
//                        Util.telemetry("glyph count", glyphCount, true);
//                        if (glyphCount == 2) {
//                            //if we have 2 glyphs
//                            myState = LEAVE_PILE;
//                        } else if (Util.rightFront.getCurrentPosition() > 400) {
//                            myState = LEAVE_PILE;
//                        }
//                    }
//                    else {
//                        myState = LEAVE_PILE;
//                    }
//                    break;
                case LEAVE_PILE:
                    //Intakes off
                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);
                    Move.strafeAngleWithoutPID(180, 0.3, 100);
                    Util.intake1.setPower(0.9);
                    Util.intake2.setPower(0.9);

                    //added without testing
                    IntakeControl.ManageGlyphCounterData();

                    Thread.sleep(100);
                    Util.telemetry("Leave_Pile!!", 0);

                    //COULD SPEED UP COMING OUT

                    //Leave pile using encoder ticks so we end up in a semi-consistent place
                    Move.rotateCounterClockwiseForMultiGlyph();
                    Move.strafeAngle(180, 0.3, 600, true);
                    Move.strafeAngle(90, 0.4, 300, true);

//                    if (!turnedRight) {
//                        //get how far we've gone and go back that far
//                        Move.strafeAngle(180, 0.3, Math.abs(Util.rightFront.
    ↪ getCurrentPosition()), true);
//                        Move.strafeAngle(90, 0.4, 400, true);
```

```
//                    } else if (turnedRight) {
//                        int currentPos = Util.rightFront.getCurrentPosition();
//                        Move.rotateCounterClockwiseForMultiGlyph();
//                        Move.strafeAngle(180, 0.3, (int) (400 + (0.6 * currentPos)), true);
//                        Move.strafeAngle(90, 0.4, (int) (400 - (0.8 * currentPos)), true);
//                    }

                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);
// else if (turnedRight && turnedLeft) {
//                        // pull out of pile with turn
//                        //retrace steps
//                        //get how far we've gone and go back that far and pull out of the
    ↪ pile (went 560 ticks in(estimated assuming degree turned is 30))
//                        encoderTicks = Util.rightFront.getCurrentPosition();
//                        Move.strafeAngle(180, 0.3, encoderTicks + 450, true); //COULD SPEED
    ↪  UP
//                        //strafe to middle for vision code
//                        Thread.sleep(100);
//                        Move.strafeAngle(90, 0.4, 100, true);
//                    }
//                    else {
//                        //pull out of pile without turn
//                        Move.strafeAngle(180, 0.35, 250, false);
//                        Move.strafeAngle(90, 0.4, 300, false);
//                    }
                    myState = BREAK_LOOP;
                    break;

            }
        }
        Util.telemetry("WHILE_Loop_broken", 0);
    }

    public static void RunWithStatesBLUE() throws InterruptedException {
        Util.resetEncoders();
        glyphCount = 0;
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        IntakeControl.ManageGlyphCounterData();
        myState = glyphState.STRAIGHT;
        turnedRight = false;
        turnedLeft = false;
        moving = false;

        while (myState != BREAK_LOOP) {
            switch (myState) {
                case STRAIGHT:
                    //Move into pile and try to collect
                    //glyphCounter(); //Get glyph count
                    if (!moving) {
                        Move.strafeAngle(0, 0.25);
                        Thread.sleep(5);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }
```

```
            IntakeControl.ManageGlyphCounterData();
            // IntakeControl.ManageDataAndHandleStalls();
            //Get glyph count

            if (glyphCount == 1){
                firstGlyphColor = myColor;
            }

            if (glyphCount == 2) {
                //if we have 2 glyphs, leave
                Util.intake1.setPower(0);
                Util.intake2.setPower(0);
                secondGlyphColor = myColor;
                //myState = LEAVE_PILE;
            }

            if (Util.rightFront.getCurrentPosition() > 400) { // cut out to improve
                ↪ proformance glyphCount == 1 ||
                //if we have gone over our encoder position go to the next collection
                myState = TURN_LEFT;
                Move.rotateCounterClockwiseForMultiGlyph();
                Util.resetEncoders();
                moving = false;
            }
            break;
        case TURN_LEFT:
            //try to collect after turning right
            //glyphCounter(); //Get glyph count
            turnedLeft = true; //we need to turn left leaving
            if (!moving) {
                Move.strafeAngleWithoutPID(0, 0.25);
                Thread.sleep(5);
                moving = true;
            } else {
                Thread.sleep(10);
            }

            IntakeControl.ManageGlyphCounterData();
            //IntakeControl.ManageDataAndHandleStalls();

            if (glyphCount == 1){
                firstGlyphColor = myColor;
            }
            if (glyphCount == 2) {
                //if we have 2 glyphs
                //myState = LEAVE_PILE;
                Util.intake1.setPower(0);
                Util.intake2.setPower(0);
                secondGlyphColor = myColor;
            }

            if (Util.rightFront.getCurrentPosition() > 400) {
                myState = LEAVE_PILE;
                // Move.rotateCounterClockwiseForMultiGlyph();
                //use this rotation if consistently only get one
                Util.telemetry("glyphs", glyphCount, true);
```

```
                    Util.resetEncoders();
                    moving = false;
                    // (hopefully by this point we have 2)
                }
                break;
//
            case LEAVE_PILE:
                //Intakes off
                Util.intake1.setPower(0);
                Util.intake2.setPower(0);
                Move.strafeAngleWithoutPID(180, 0.3, 100);
                Util.intake1.setPower(0.9);
                Util.intake2.setPower(0.9);
                Thread.sleep(100);
                Util.telemetry("Leave_Pile!!", 0);

                //COULD SPEED UP COMING OUT

                //Leave pile using encoder ticks so we end up in a semi-consistent place
                Move.rotateClockwiseForMultiGlyph();
                Move.strafeAngle(180, 0.3, 600, true);
                Move.strafeAngle(-90, 0.4, 150, true);

//                  if (!turnedRight) {
//                      //get how far we've gone and go back that far
//                      Move.strafeAngle(180, 0.3, Math.abs(Util.rightFront.
    ↪ getCurrentPosition()), true);
//                      Move.strafeAngle(90, 0.4, 400, true);
//                  } else if (turnedRight) {
//                      int currentPos = Util.rightFront.getCurrentPosition();
//                      Move.rotateCounterClockwiseForMultiGlyph();
//                      Move.strafeAngle(180, 0.3, (int) (400 + (0.6 * currentPos)), true);
//                      Move.strafeAngle(90, 0.4, (int) (400 - (0.8 * currentPos)), true);
//                  }

                Util.intake1.setPower(0);
                Util.intake2.setPower(0);
// else if (turnedRight && turnedLeft) {
//                      // pull out of pile with turn
//                      //retrace steps
//                      //get how far we've gone and go back that far and pull out of the
    ↪ pile (went 560 ticks in(estimated assuming degree turned is 30))
//                      encoderTicks = Util.rightFront.getCurrentPosition();
//                      Move.strafeAngle(180, 0.3, encoderTicks + 450, true); //COULD SPEED
    ↪  UP
//                      //strafe to middle for vision code
//                      Thread.sleep(100);
//                      Move.strafeAngle(90, 0.4, 100, true);
//                  }
//                  else {
//                      //pull out of pile without turn
//                      Move.strafeAngle(180, 0.35, 250, false);
//                      Move.strafeAngle(90, 0.4, 300, false);
//                  }
                myState = BREAK_LOOP;
                break;
```

```
            }
        }
        Util.telemetry("WHILE_Loop_broken", 0);
    }

    public static void RunWithStatesFarRed() throws InterruptedException{
        //Move.strafeAngleWithHeading(90,0.4 ,600,true, -88);
        Move.strafeAngleWithHeading(0,0.5, 550,false, -88);
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        Move.strafeAngleWithHeading(0,0.4, 400,true, -88);
//
        Util.resetEncoders();
        glyphCount = 0;
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        IntakeControl.ManageGlyphCounterData();
        myState = glyphState.STRAIGHT;
        turnedRight = false;
        turnedLeft = false;
        moving = false;

        while (myState != BREAK_LOOP) {
            switch (myState) {
                case STRAIGHT:
                    //Move into pile and try to collect
                    //glyphCounter(); //Get glyph count
                    if (!moving) {
                        Move.strafeAngleWithHeading(0, 0.25, -88);
                        Thread.sleep(5);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }

                    IntakeControl.ManageGlyphCounterData();
                    // IntakeControl.ManageDataAndHandleStalls();
                    //Get glyph count

                    if (glyphCount == 2) {
                        //if we have 2 glyphs, leave
                        Util.intake1.setPower(0);
                        Util.intake2.setPower(0);
                        //myState = LEAVE_PILE;
                    }

                    if (Util.rightFront.getCurrentPosition() > 100) { // cut out to improve
                        ↪ proformance glyphCount == 1 ||
                        //if we have gone over our encoder position go to the next collection
                        myState = TURN_RIGHT;
                        Move.rotateClockwiseForMultiGlyph();
                        Util.resetEncoders();
                        moving = false;
                    }
                    break;
                case TURN_RIGHT:
                    //try to collect after turning right
```

```
                    //glyphCounter(); //Get glyph count
                    turnedRight = true; //we need to turn left leaving
                    if (!moving) {
                        Move.strafeAngleWithoutPID(0, 0.25);
                        Thread.sleep(5);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }

                    IntakeControl.ManageGlyphCounterData();
                    //IntakeControl.ManageDataAndHandleStalls();

                    if (glyphCount == 2) {
                        //if we have 2 glyphs
                        //myState = LEAVE_PILE;
                        Util.intake1.setPower(0);
                        Util.intake2.setPower(0);
                    }

                    if (Util.rightFront.getCurrentPosition() > 550) {
                        myState = LEAVE_PILE;
                        // Move.rotateCounterClockwiseForMultiGlyph();
                        //use this rotation if consistently only get one
                        Util.telemetry("glyphs", glyphCount, true);
                        Util.resetEncoders();
                        moving = false;
                        // (hopefully by this point we have 2)
                    }
                    break;
//                case TURN_LEFT:
//                    //glyphCounter(); //Get glyph count
//                    if (VuforiaGoToColumn.column != VuforiaGoToColumn.columnState.LEFT){
//                        Util.telemetry("Turn Left!!", 0);
//                        turnedLeft = true; //used to help with leaving pile
//                        //try to collect after turning left
//                        if (!moving) {
//                            Move.strafeAngleWithoutPID(0, 0.25);
//                            moving = true;
//                        } else {
//                            Thread.sleep(10);
//                        }
//                        IntakeControl.ManageGlyphCounterData();
//                        IntakeControl.ManageDataAndHandleStalls();
//
//
//                        Util.telemetry("glyph count", glyphCount, true);
//                        if (glyphCount == 2) {
//                            //if we have 2 glyphs
//                            myState = LEAVE_PILE;
//                        } else if (Util.rightFront.getCurrentPosition() > 400) {
//                            myState = LEAVE_PILE;
//                        }
//                    }
//                    else {
//                        myState = LEAVE_PILE;
//                    }
```

```
//                    break;
                case LEAVE_PILE:
                    //Intakes off
                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);
                    Move.strafeAngleWithoutPID(180, 0.3, 550);
                    Util.setAllPowers(0);
                    Move.rotateCounterClockwiseForMultiGlyph();
                    Util.setAllPowers(0);
                    Util.intake1.setPower(0.9);
                    Util.intake2.setPower(0.9);
                    Move.strafeAngleWithHeading(-90, 0.4, 250,false, -88);
                    Move.strafeAngleforTimeWithHeading(-90, 0.35, 0.35, -88);
                    Util.telemetry("Leave_Pile!!", 0);


                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);

                    myState = BREAK_LOOP;
                    break;

            }
        }
        Util.telemetry("WHILE_Loop_broken", 0);
    }
    public static void RunWithStatesFarBlue() throws InterruptedException{
        //Move.strafeAngleWithHeading(90,0.4 ,600,true, -88);
        Move.strafeAngleWithHeading(0,0.5, 550,false, 88);
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        Move.strafeAngleWithHeading(0,0.4, 400,true, 88);
//
        Util.resetEncoders();
        glyphCount = 0;
        Util.intake1.setPower(0.9); //trying .8 may give counter longer
        Util.intake2.setPower(0.9);
        IntakeControl.ManageGlyphCounterData();
        myState = glyphState.STRAIGHT;
        turnedRight = false;
        turnedLeft = false;
        moving = false;

        while (myState != BREAK_LOOP) {
            switch (myState) {
                case STRAIGHT:
                    //Move into pile and try to collect
                    //glyphCounter(); //Get glyph count
                    if (!moving) {
                        Move.strafeAngleWithHeading(0, 0.25, 88);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }

                    IntakeControl.ManageGlyphCounterData();
                    // IntakeControl.ManageDataAndHandleStalls();
```

```
                    //Get glyph count

                    if (glyphCount == 2) {
                        //if we have 2 glyphs, leave
                        Util.intake1.setPower(0);
                        Util.intake2.setPower(0);
                        //myState = LEAVE_PILE;
                    }

                    if (Util.leftFront.getCurrentPosition() > 100) { // cut out to improve
                        ↪ proformance glyphCount == 1 ||
                        //if we have gone over our encoder position go to the next collection
                        myState = TURN_RIGHT;
                        Move.rotateCounterClockwiseForMultiGlyph();
                        Util.resetEncoders();
                        moving = false;
                    }
                    break;
                case TURN_RIGHT:
                    //try to collect after turning right
                    //glyphCounter(); //Get glyph count
                    turnedRight = true; //we need to turn left leaving
                    if (!moving) {
                        Move.strafeAngleWithoutPID(0, 0.25);
                        moving = true;
                    } else {
                        Thread.sleep(10);
                    }

                    IntakeControl.ManageGlyphCounterData();
                    //IntakeControl.ManageDataAndHandleStalls();

                    if (glyphCount == 2) {
                        //if we have 2 glyphs
                        //myState = LEAVE_PILE;
                        Util.intake1.setPower(0);
                        Util.intake2.setPower(0);
                    }

                    if (Util.rightFront.getCurrentPosition() > 100) { //needs to be 550
                        myState = LEAVE_PILE;
                        // Move.rotateCounterClockwiseForMultiGlyph();
                        //use this rotation if consistently only get one
                        Util.telemetry("glyphs", glyphCount, true);
                        Util.resetEncoders();
                        moving = false;
                        // (hopefully by this point we have 2)
                    }
                    break;
//                case TURN_LEFT:
//                    //glyphCounter(); //Get glyph count
//                    if (VuforiaGoToColumn.column != VuforiaGoToColumn.columnState.LEFT){
//                        Util.telemetry("Turn Left!!", 0);
//                        turnedLeft = true; //used to help with leaving pile
//                        //try to collect after turning left
//                        if (!moving) {
//                            Move.strafeAngleWithoutPID(0, 0.25);
```

```
//                          moving = true;
//                      } else {
//                          Thread.sleep(10);
//                      }
//                      IntakeControl.ManageGlyphCounterData();
//                      IntakeControl.ManageDataAndHandleStalls();
//
//
//                      Util.telemetry("glyph count", glyphCount, true);
//                      if (glyphCount == 2) {
//                          //if we have 2 glyphs
//                          myState = LEAVE_PILE;
//                      } else if (Util.rightFront.getCurrentPosition() > 400) {
//                          myState = LEAVE_PILE;
//                      }
//                  }
//                  else {
//                      myState = LEAVE_PILE;
//                  }
//                  break;
                case LEAVE_PILE:
                    //Intakes off
                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);
                    Move.strafeAngleWithoutPID(180, 0.3, 100); //needs to be 550
                    Util.setAllPowers(0);
                    Move.rotateClockwiseForMultiGlyph();
                    Util.setAllPowers(0);
                    Util.intake1.setPower(0.9);
                    Util.intake2.setPower(0.9);
                    Move.strafeAngleWithHeading(90, 0.4, 250,false, 88);
                    Move.strafeAngleforTimeWithHeading(90, 0.35, 0.35, 88);
                    Util.telemetry("Leave_Pile!!", 0);


                    Util.intake1.setPower(0);
                    Util.intake2.setPower(0);

                    myState = BREAK_LOOP;
                    break;

            }
        }
        Util.telemetry("WHILE_Loop_broken", 0);
    }
}
```

goToColumnBasedOnMultiGlyphs:

```
package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.LinearOpMode;

import org.firstinspires.ftc.teamcode.Util;

/**
 * Created by elliot on 4/11/18.
 */

public class goToColumnBasedOnMultiGlyphs {

    public static VuforiaGoToColumn.columnState NearRed (VuforiaGoToColumn.columnState column
        ↪ , LinearOpMode opMode) throws InterruptedException {
        if (getGlyphs.glyphCount == 1){
            if (getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN){
                Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
                Util.liftTrayforMulti = true;
                return column;
            }
            else {
                Util.telemetry("SNAKE_CYPHER", 0);
                if (column != VuforiaGoToColumn.columnState.CENTER){
                    Util.liftTrayforMulti = true;
                    return column;
                }
                else {
                   return VuforiaGoToColumn.columnState.LEFT;
                }
            }
        }
        else if (getGlyphs.glyphCount == 2){
            if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN) && (getGlyphs.
                ↪ secondGlyphColor == getGlyphs.glyphColor.BROWN)){
                Util.telemetry("SNAKE_CYPHER", 0);
                if (column == VuforiaGoToColumn.columnState.RIGHT){
                    return VuforiaGoToColumn.columnState.LEFT;
                }
                else if (column == VuforiaGoToColumn.columnState.CENTER){
                    return VuforiaGoToColumn.columnState.RIGHT;

                }
                else if (column == VuforiaGoToColumn.columnState.LEFT){
                    return VuforiaGoToColumn.columnState.RIGHT;

                }
            }
            else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.GREY) && (getGlyphs.
                ↪ secondGlyphColor == getGlyphs.glyphColor.GREY)){
                Util.telemetry("SNAKE_CYPHER", 0);
                if (column == VuforiaGoToColumn.columnState.RIGHT){
                    Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                    return VuforiaGoToColumn.columnState.LEFT;
                }
                else if (column == VuforiaGoToColumn.columnState.CENTER){
                    return VuforiaGoToColumn.columnState.LEFT;
```

```
            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                return VuforiaGoToColumn.columnState.RIGHT;
            }
        }
        else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.GREY) && (getGlyphs.
            ↪ secondGlyphColor == getGlyphs.glyphColor.BROWN)){
            Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
            if (column == VuforiaGoToColumn.columnState.RIGHT){
                return VuforiaGoToColumn.columnState.LEFT;
            }
            else if (column == VuforiaGoToColumn.columnState.CENTER){
                Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                return VuforiaGoToColumn.columnState.RIGHT;

            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                return VuforiaGoToColumn.columnState.RIGHT;

            }
        }
        else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN) && (getGlyphs.
            ↪ secondGlyphColor == getGlyphs.glyphColor.GREY)){
            Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
            if (column == VuforiaGoToColumn.columnState.RIGHT){
                return VuforiaGoToColumn.columnState.CENTER;
            }
            else if (column == VuforiaGoToColumn.columnState.CENTER){
                return VuforiaGoToColumn.columnState.RIGHT;

            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                return VuforiaGoToColumn.columnState.CENTER;

            }
        }
        else {
            if (column == VuforiaGoToColumn.columnState.RIGHT){
                return VuforiaGoToColumn.columnState.LEFT;
            }
            else if (column == VuforiaGoToColumn.columnState.CENTER){
                return VuforiaGoToColumn.columnState.LEFT;

            }
            else if (column == VuforiaGoToColumn.columnState.LEFT){
                return VuforiaGoToColumn.columnState.RIGHT;

            }
        }
    }
    else {
        return VuforiaGoToColumn.columnState.UNKNOWN;
    }

    return VuforiaGoToColumn.columnState.UNKNOWN;
}
```

```
public static VuforiaGoToColumn.columnState NearBlue (VuforiaGoToColumn.columnState
    ↪ column, LinearOpMode opMode) throws InterruptedException {
    if (getGlyphs.glyphCount == 1) {
        if (getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN) {
            Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
            Util.liftTrayforMulti = true;
            return column;
        } else {
            Util.telemetry("SNAKE_CYPHER", 0);
            if (column != VuforiaGoToColumn.columnState.CENTER) {
                Util.liftTrayforMulti = true;
                return column;
            } else {
                return VuforiaGoToColumn.columnState.LEFT;
            }
        }
    } else if (getGlyphs.glyphCount == 2) {
        if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN) && (getGlyphs.
            ↪ secondGlyphColor == getGlyphs.glyphColor.BROWN)) {
            Util.telemetry("SNAKE_CYPHER", 0);
            if (column == VuforiaGoToColumn.columnState.RIGHT) {
                return VuforiaGoToColumn.columnState.LEFT;
            } else if (column == VuforiaGoToColumn.columnState.CENTER) {
                return VuforiaGoToColumn.columnState.RIGHT;

            } else if (column == VuforiaGoToColumn.columnState.LEFT) {
                return VuforiaGoToColumn.columnState.RIGHT;

            }
        } else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.GREY) && (getGlyphs
            ↪ .secondGlyphColor == getGlyphs.glyphColor.GREY)) {
            Util.telemetry("SNAKE_CYPHER", 0);
            if (column == VuforiaGoToColumn.columnState.RIGHT) {
                Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                return VuforiaGoToColumn.columnState.LEFT;
            } else if (column == VuforiaGoToColumn.columnState.CENTER) {
                return VuforiaGoToColumn.columnState.LEFT;
            } else if (column == VuforiaGoToColumn.columnState.LEFT) {
                Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                return VuforiaGoToColumn.columnState.RIGHT;
            }
        } else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.GREY) && (getGlyphs
            ↪ .secondGlyphColor == getGlyphs.glyphColor.BROWN)) {
            Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
            if (column == VuforiaGoToColumn.columnState.RIGHT) {
                return VuforiaGoToColumn.columnState.LEFT;
            } else if (column == VuforiaGoToColumn.columnState.CENTER) {
                Util.telemetry("REMOVE_FIRST_GLYPH", 0);
                return VuforiaGoToColumn.columnState.LEFT;

            } else if (column == VuforiaGoToColumn.columnState.LEFT) {
                return VuforiaGoToColumn.columnState.RIGHT;

            }
        } else if ((getGlyphs.firstGlyphColor == getGlyphs.glyphColor.BROWN) && (
            ↪ getGlyphs.secondGlyphColor == getGlyphs.glyphColor.GREY)) {
```

```
                Util.telemetry("CHEKERBOARD/FROG_CYPHER", 0);
                if (column == VuforiaGoToColumn.columnState.RIGHT) {
                    return VuforiaGoToColumn.columnState.CENTER;

                } else if (column == VuforiaGoToColumn.columnState.CENTER) {
                    return VuforiaGoToColumn.columnState.LEFT;

                } else if (column == VuforiaGoToColumn.columnState.LEFT) {
                    return VuforiaGoToColumn.columnState.CENTER;

                }
            } else {
                if (column == VuforiaGoToColumn.columnState.RIGHT) {
                    return VuforiaGoToColumn.columnState.LEFT;

                } else if (column == VuforiaGoToColumn.columnState.CENTER) {
                    return VuforiaGoToColumn.columnState.LEFT;

                } else if (column == VuforiaGoToColumn.columnState.LEFT) {
                    return VuforiaGoToColumn.columnState.RIGHT;

                }
            }
        } else {
            Util.telemetry("COLUMN_UNKNOWN", 0);
            return VuforiaGoToColumn.columnState.UNKNOWN;
        }

        return VuforiaGoToColumn.columnState.UNKNOWN;
    }
}
```

Intake Control class:

```
package org.firstinspires.ftc.teamcode.RelicRecovery;

import com.qualcomm.robotcore.eventloop.opmode.TeleOp;

import org.firstinspires.ftc.robotcore.external.navigation.DistanceUnit;
import org.firstinspires.ftc.teamcode.Util;

import java.util.LinkedList;
import java.util.Queue;

/**
 * Created by elliot on 3/17/18.
 */

public class IntakeControl {
    static int red;
    static int green;
    static int blue;
    static int alpha;
    static double distance;

    static int rightIntakePos;
    static int leftIntakePos;
    static int lastRightIntakePos;
    static int lastLeftIntakePos;

    static double rightIntakeDiff;
    static double leftIntakeDiff;

    static double deltaRight;
    static double deltaLeft;

    static double lowestDeltaRight;
    static double lowestDeltaLeft;

    static double rightIntakeSum;
    static double leftIntakeSum;

    static double MOVING_AVERAGE_LENGTH = 20, MEASURING_INTERVAL = 10;

    static boolean canCount;

    static Queue<Double> rightIntakeQueue, leftIntakeQueue;
    static boolean queueClear;
    static boolean queueFill;

    static long timeSum;

    static long currentTime;
    static long oldTime;

    public static void init() throws InterruptedException {
        Util.resetEncoders();
        fillQueue();
        getGlyphs.glyphCount = 0;
        canCount = true;
```

```
    lowestDeltaLeft = 30;
    lowestDeltaRight = 30;
    oldTime = System.nanoTime() / 1000000;
}

public static void teleOpinit() {
    canCount = true;
}

public static void ManageEncoderData(double elapsedTime) throws InterruptedException {
    rightIntakePos = Util.intake2.getCurrentPosition();
    leftIntakePos = Util.intake1.getCurrentPosition();

    rightIntakeDiff = Math.abs(rightIntakePos - lastRightIntakePos);
    leftIntakeDiff = Math.abs(leftIntakePos - lastLeftIntakePos);

    rightIntakeDiff = (MEASURING_INTERVAL / elapsedTime) * rightIntakeDiff;
    leftIntakeDiff = (MEASURING_INTERVAL / elapsedTime) * leftIntakeDiff;

    rightIntakeSum = rightIntakeSum + rightIntakeDiff - rightIntakeQueue.poll();
    rightIntakeQueue.add(rightIntakeDiff);
    leftIntakeSum = leftIntakeSum + leftIntakeDiff - leftIntakeQueue.poll();
    leftIntakeQueue.add(leftIntakeDiff);

    lastRightIntakePos = rightIntakePos;
    lastLeftIntakePos = leftIntakePos;

    queueClear = false;
    queueFill = false;
}

public static void handleStalls() throws InterruptedException {
    //Handle Stalls
    deltaRight = rightIntakeSum / MOVING_AVERAGE_LENGTH;
    deltaLeft = leftIntakeSum / MOVING_AVERAGE_LENGTH;

    if (deltaRight < lowestDeltaRight){
        lowestDeltaRight = deltaRight;
    }
    if (deltaLeft < lowestDeltaLeft){
        lowestDeltaLeft = deltaLeft;
    }

    Util.telemetry("Lowest_right_delta", lowestDeltaRight);
    Util.telemetry("Lowest_left_delta", lowestDeltaLeft);

    if ((deltaRight < 8 || deltaLeft < 8)) {
        Util.intake1.setPower(-0.9);
        Util.intake2.setPower(-0.9);
        Thread.sleep(500);
        Util.intake1.setPower(0.9);
        Util.intake2.setPower(0.9);
        fillQueue();
    }

}
```

```
    public static void ManageDataAndHandleStalls() throws InterruptedException {
        currentTime = System.nanoTime() / 1000000;
        IntakeControl.ManageEncoderData(currentTime - oldTime);
        oldTime = currentTime;
        IntakeControl.handleStalls();
       // Thread.sleep(10);
    }

    public static void clearQueue() {
        if (queueClear) return;

        rightIntakeQueue = new LinkedList<>();
        leftIntakeQueue = new LinkedList<>();

        rightIntakeSum = 0;
        leftIntakeSum = 0;

        for (int i = 0; i < MOVING_AVERAGE_LENGTH; i++){
            rightIntakeQueue.add(0.0);
            leftIntakeQueue.add(0.0);
        }

        queueClear = true;
    }

    public static void fillQueue() {
        if (queueFill) return;

        rightIntakeQueue = new LinkedList<>();
        leftIntakeQueue = new LinkedList<>();

        rightIntakeSum = 500; //change to reflect limit on the deltas
        leftIntakeSum = 500;

        for (int i = 0; i < MOVING_AVERAGE_LENGTH; i++){
            rightIntakeQueue.add(30.0);
            leftIntakeQueue.add(30.0);
        }

        queueFill = true;
    }

    public static void ManageGlyphCounterData() {
        distance = Util.glyphCounterDistance.getDistance(DistanceUnit.CM);

//        if (Double.isNaN(distance)){
//            distance = 100;
//        }


        if (Double.isNaN(distance)){
            canCount = true;
            //Util.telemetry("Can Count!", true);
        }

        if ((!Double.isNaN(distance)) && canCount){
            //Util.telemetry("counting!", true);
```

```
            getGlyphs.glyphCount += 1;
            red = Util.glyphCounterColor.red();
            blue = Util.glyphCounterColor.green();
            green = Util.glyphCounterColor.blue();
            alpha = Util.glyphCounterColor.alpha();

            if (red > 100){
                red = 0;
            }
            if (green > 100){
                green = 0;
            }
            if (blue > 100){
                blue = 0;
            }
            if (alpha > 250){
                alpha = 0;
            }

            if (red >= 25 || green >= 20 || blue >= 20 || alpha >= 60){
                getGlyphs.myColor = getGlyphs.glyphColor.GREY;
                Util.telemetry("glyph_color" , getGlyphs.myColor, true);
            }
            else if (((0 < red) && (red < 25)) && ((0 < green) && (green < 20)) && ((0 < blue
                ↪ ) && (blue < 20)) && ((0 < alpha) && (alpha < 60))){
                getGlyphs.myColor = getGlyphs.glyphColor.BROWN;
                Util.telemetry("glyph_color" , getGlyphs.myColor, true);
            }
            else {
                getGlyphs.myColor = getGlyphs.glyphColor.UNKNOWN;
                Util.telemetry("glyph_color" , getGlyphs.myColor, true);
            }
            Util.telemetry("glyph_number", getGlyphs.glyphCount, true);

            canCount = false;
            //Util.telemetry("Distance", distance, true);
        }
    }

    public static void ManageGlyphCounterDataTeleOp() {
        distance = MecanumTeleop.glyphCounterDistance.getDistance(DistanceUnit.CM);

//        if (Double.isNaN(distance)){
//            distance = 100;
//        }


        if (Double.isNaN(distance)){
            canCount = true;
            //Util.telemetry("Can Count!", true);
        }

        if (!(Double.isNaN(distance)) && canCount){
            //Util.telemetry("counting!", true);
            getGlyphs.glyphCount += 1;
            red = MecanumTeleop.glyphCounterColor.red();
            blue = MecanumTeleop.glyphCounterColor.green();
```

```
            green = MecanumTeleop.glyphCounterColor.blue();
            alpha = MecanumTeleop.glyphCounterColor.alpha();

            if (red > 100){
                red = 0;
            }
            if (green > 100){
                green = 0;
            }
            if (blue > 100){
                blue = 0;
            }
            if (alpha > 250){
                alpha = 0;
            }

            if (red >= 25 || green >= 20 || blue >= 20 || alpha >= 60){
                getGlyphs.myColor = getGlyphs.glyphColor.GREY;
            }
            else if (((0 < red) && (red < 25)) && ((0 < green) && (green < 20)) && ((0 < blue
                ↪ ) && (blue < 20)) && ((0 < alpha) && (alpha < 60))){
                getGlyphs.myColor = getGlyphs.glyphColor.BROWN;
            }
            else {
                getGlyphs.myColor = getGlyphs.glyphColor.UNKNOWN;
            }

            canCount = false;
            //Util.telemetry("Distance", distance, true);
        }
    }
}
```