```c
#include "catapult.h"

const int OFF = 0;
const int FULL_POWER = 127;

//Minimum value from joystick analogs needed to activate drive wheels
int deadzone = 25;

//Power used by rotate function
int rotatePower = 25;

//Power used by mechanumDrive function
int mechanumPower = FULL_POWER;

//Quad values used by the armPosition and gatePosition tasks
int loadStop = 7;
int fireStop = 55;
int gateCloseQuad = -80;
int gateOpenQuad = -10;

//Motor Powers for Arm + Gate
int gateHoldPower = 15;
int gateDrivePower = 90;
int armHoldPower = 15;
int armLoadPower = FULL_POWER;
int shortPower = 40;
int midPower = 50;
int tilePower = 81;
int cornerPower = 90;

//Variable used to know when the bot is participating in a skills event or a normal match
bool skills = false;

//Variables used to know firing distance and gate/arm status
Distance distance = SHORT;
ArmStatus armStatus = ARM_LOAD;
GateStatus gateStatus = GATE_CLOSE;

//Variables used to turn on/off the gate and arm monitoring tasks
int ballCount = 0;
bool override = false;
bool armRunning = false;
bool gateRunning = false;
```

```
//Stuff to do before autonomous
void pre_auton(){

}

/*
 ######   #######   #####   #######
 #     #  #        #     #     #
 #     #  #        #           #
 ######   #####     #####      #
 #   #    #              #     #
 #    #   #        #     #     #
 #     #  #######   #####      #

 #######  #######
 #     #  #
 #     #  #
 #     #  #####
 #     #  #
 #     #  #
 #######  #

 #######  #     #  #     #   #####   #######  ###  #######  #     #   #####
 #        #     #  ##    #  #     #     #      #   #     #  ##    #  #     #
 #        #     #  # #   #  #           #      #   #     #  # #   #  #
 #####    #     #  #  #  #  #           #      #   #     #  #  #  #   #####
 #        #     #  #   # #  #           #      #   #     #  #   # #        #
 #        #     #  #    ##  #     #     #      #   #     #  #    ##  #     #
 #         #####   #     #   #####      #     ###  #######  #     #   #####

 ######   #######   #####    #####   ######   ###  ######   #######  ######
 #     #  #        #     #  #     #  #     #   #   #     #  #        #     #
 #     #  #        #        #        #     #   #   #     #  #        #     #
 #     #  #####     #####   #        ######    #   ######   #####    #     #
 #     #  #              #  #        #   #     #   #     #  #        #     #
 #     #  #        #     #  #     #  #    #    #   #     #  #        #     #
 ######   #######   #####    #####   #     #  ###  ######   #######  ######

 ###  #     #
  #   ##    #
  #   # #   #
  #   #  #  #
  #   #   # #
  #   #    ##
```

```
 ###  #     #

 #     #  #######     #     ######   #######  ######
 #     #  #          # #    #     #  #        #     #
 #     #  #         #   #   #     #  #        #     #
 #######  #####    #     #  #     #  #####    ######
 #     #  #        #######  #     #  #        #   #
 #     #  #        #     #  #     #  #        #    #
 #     #  #######  #     #  ######   #######  #     #

 #######  ###  #        #######
 #         #   #        #
 #         #   #        #
 #####     #   #        #####
 #         #   #        #
 #         #   #        #
 #        ###  #######  #######
*/

void rotate(bool left){
        if(left){
                motor[backRight] = rotatePower;
                motor[backLeft] = -rotatePower;
                motor[frontLeft] = -rotatePower;
                motor[frontRight] = rotatePower;
        }
        else{
                motor[backRight] = -rotatePower;
                motor[backLeft] = rotatePower;
                motor[frontLeft] = rotatePower;
                motor[frontRight] = -mechanumPower;
        }
}

void mechanumDrive(bool left){
        if(left){
                motor[backRight] = -mechanumPower;
                motor[backLeft] = mechanumPower;
                motor[frontLeft] = -mechanumPower;
                motor[frontRight] = mechanumPower;
        }
        else{
                motor[backRight] = mechanumPower;
                motor[backLeft] = -mechanumPower;
```

```
                motor[frontLeft] = mechanumPower;
                motor[frontRight] = -mechanumPower;
        }
}

void drive(bool leftSide, bool backwards, int speed){
        if(backwards)
                speed = -speed;

        if(leftSide){
                motor[frontLeft] = speed;
                motor[backLeft] = speed;
        }
        else{
                motor[frontRight] = speed;
                motor[backRight] = speed;
        }
}

void resetSensors(){
        SensorValue[armQuad] = OFF;
        SensorValue[gateQuad] = OFF;
}

void stopLeftDrive(){
        motor[backLeft] = OFF;
        motor[frontLeft] = OFF;
}

void stopRightDrive(){
        motor[backRight] = OFF;
        motor[frontRight] = OFF;
}

void stopAllDrive(){
        motor[backRight] = OFF;
        motor[frontRight] = OFF;
        motor[backLeft] = OFF;
        motor[frontLeft] = OFF;
}

void tankDrive(){
        if(abs(vexRT[Ch2]) > deadzone){
                motor[backRight] = vexRT[Ch2];
```

```
                motor[frontRight] = vexRT[Ch2];
        }
        else{
                stopRightDrive();
        }

        if(abs(vexRT[Ch3]) > deadzone){
                motor[frontLeft] = vexRT[Ch3];
                motor[backLeft] = vexRT[Ch3];
        }
        else{
                stopLeftDrive();
        }
}

void fire(bool load, int speed){
        if(load){
                motor[yOne] = -speed;
                motor[yTwo] = -speed;
                motor[yThree] = -speed;
        }
        else{
                motor[yOne] = speed;
                motor[yTwo] = speed;
                motor[yThree] = speed;
        }
}

void stopArm(){
        motor[yOne] = OFF;
        motor[yTwo] = OFF;
        motor[yThree] = OFF;
}

void setArm(ArmStatus status){
        armStatus = status;
}

void setDistance(Distance newDistance){
        distance = newDistance;
}

int getArmSpeed(){
        int speed;
```

```
	switch(distance){
	case SHORT:
		speed = shortPower;
		break;

	case MID:
		speed = midPower;
		break;

	case TILE:
		speed = tilePower;
		break;

	case CORNER:
		speed = cornerPower;
		break;

	default:
		speed = OFF;
		break;
	}
	return speed;
}

void setGate(GateStatus status){
	gateStatus = status;
	gateRunning = true;
}

task armPosition(){
	while(true){
	if(armStatus == ARM_FIRE){
			armRunning = true;
		}
		while(armRunning){
			if(armStatus == ARM_FIRE && (SensorValue[armSwitch] == 0 || override)){

				//Open gate to start loading of another ball
				setGate(GATE_OPEN);

				int speed = getArmSpeed();
				if(skills) wait1Msec(100);
			UP:
				clearTimer(T1);
```

```
                                //Fire currently loaded ball
                                while(abs(SensorValue[armQuad]) < fireStop){
                                        if(time1[T1] > 1250){
                                                goto DOWN;
                                        }
                                        if(speed == OFF){
                                                break;
                                        }
                                        fire(false, speed);
                                }
                                stopArm();

                                wait1Msec(50);

                                //Bring arm back down
                        DOWN:
                                clearTimer(T1);
                                while(abs(SensorValue[armQuad]) > loadStop){
                                        if(time1[T1] > 1250){
                                                goto UP;
                                        }
                                        fire(true, armLoadPower);
                                }
                                fire(true, armHoldPower);

                                //Set arm to LOADED state
                                setArm(ARM_LOAD);
                                armRunning = false;
                                override = false;
                        }
                        else{
                                setArm(ARM_LOAD);
                                armRunning = false;
                                override = false;
                        }
                }
        }
}

task gatePosition(){
        while(true){

                if(SensorValue[gateSwitch] == 0){
                        setGate(GATE_CLOSE);
```

```
                }

                while(gateRunning){

                        //Open gate and add one to ballCount
                        if(gateStatus == GATE_OPEN){
                                ballCount++;
                                while(SensorValue[gateQuad] < gateOpenQuad){
                                        motor[gate] = -gateDrivePower;
                                }
                                motor[gate] = -gateHoldPower;
                                gateRunning = false;
                        }

                        //Close gate
                        if(gateStatus == GATE_CLOSE){
                                while(SensorValue[gateQuad] > gateCloseQuad){
                                        motor[gate] = gateDrivePower;
                                }
                                motor[gate] = gateHoldPower;
                                gateRunning = false;
                        }
                }
        }
}
```