```cpp
/*---------------------------------------------------------------*/
/*    5249Z-Ignite                                               */
/*    Version: 1.2.0                                             */
/*    File: main.cpp                                             */
/*    Description: Main control file for the program, contains  */
/*      main.cpp                                                 */
/*---------------------------------------------------------------*/
#include "RobotConfig.h"

static int mode = -1;//Mode for the robot to operate in

void calibrateGyros(){//Calibrates gyros
    ctrPrimary.Screen.clearScreen();
    ctrPrimary.Screen.print("Calibrating");
    navInert.calibrate();
    //gyroDrive.calibrate();
    task::sleep(3000);
    ctrPrimary.Screen.clearScreen();
}
void stopAllMotors(){//stops all motors on the robot
    mtrLeft.stop(brakeType::coast);
    mtrRight.stop(brakeType::coast);
    mtrLeftFront.stop(brakeType::coast);
    mtrRightFront.stop(brakeType::coast);
    mtrArm.stop(brakeType::coast);
    mtrIntakeLeft.stop(brakeType::coast);
    mtrIntakeRight.stop(brakeType::coast);
    mtrRampLift.stop(brakeType::coast);
    task::stopAll();
}
void clearMotorRotations(){
    mtrLeft.resetRotation();
    mtrRight.resetRotation();
    mtrLeftFront.resetRotation();
    mtrRightFront.resetRotation();
    mtrArm.resetRotation();
    mtrIntakeLeft.resetRotation();
    mtrIntakeRight.resetRotation();
    mtrRampLift.resetRotation();
    mtrLeft.setMaxTorque(100, percentUnits::pct);
    mtrRight.setMaxTorque(100, percentUnits::pct);
    mtrLeftFront.setMaxTorque(100, percentUnits::pct);
    mtrRightFront.setMaxTorque(100, percentUnits::pct);
    task::sleep(500);
}
bool isField(){//Method for checking if either field control device is connected
    return compControl.isCompetitionSwitch() || compControl.isFieldControl();
}
class DisplaySelection {//Class created to hold and change the values needed to move
the display up and down
        private:
            int maxLines = 3;//Number of controller display lines
            int topLine = 0;//Choice on the top line of the controller
            int position = 0;//Position of the arrow
            unsigned int max = 0;//Max number of choices

            int getCurrent(){//returns the option the arrow is on
                return topLine + position;
            }
            void moveDown(){//Moves display down
                if (getCurrent() != max - 1){//If the arrow is not at the last choice,
                move everything down
                    if (position == maxLines - 1){//Move the options down if the arrow
                    is at the bottom
                        topLine ++;
                    } else {//Move the arrow down otherwise
                        position ++;
                    }
                } else {//If the arrow is at the last choice, return to the top
                    topLine = 0;
                    position = 0;
                }
            }
            void moveUp(){//Moves Display up
                if (getCurrent() != 0){//If the arrow is at not at the first selection, move everything up
                    if (position == 0){//move the options up if the arrow is at the top
                        topLine --;
                    } else {//Otherwise move the arrow up
                        position --;
                    }
                } else {//If the arrow is at the first choice, go to the bottom
                    position = maxLines - 1;
                    topLine = max - maxLines;
                }
            }
        public:
            char text[8][32];//storage for text options
            DisplaySelection(unsigned int maxOptions){//Constructor
                if (maxOptions < maxLines){//Sets the maxlines to the option number in case there are less options that usable lines
                    maxLines = maxOptions;
                }
                max = maxOptions;//Set the max number of options
            }
            int select(){//returns the chosen selection
                while(true){//repeat update until a selection is chosen
                    if(ctrPrimary.ButtonA.pressing()){//Return the current number if a selection has been made
                        while(ctrPrimary.ButtonA.pressing() || ctrPrimary.ButtonUp.pressing() || ctrPrimary.ButtonDown.pressing()){wait(20);}
                        return getCurrent();
                    }
                    if(ctrPrimary.ButtonUp.pressing()){//Move up if up button is pressed
                        moveUp();
                    }
                    if(ctrPrimary.ButtonDown.pressing()){//Move down if down button is pressed
                        moveDown();
                    }
                    ctrPrimary.Screen.clearScreen();//clears the screen
                    for (int i=0; i < maxLines; i++){//Displays lines of text based on instance variables
                        ctrPrimary.Screen.setCursor(i+1,3);//
                        ctrPrimary.Screen.print("%s", text[i + topLine]);
                    }
                    ctrPrimary.Screen.setCursor(position+1,0);
                    ctrPrimary.Screen.print("->");//Print the arrow at the position
                    while(ctrPrimary.ButtonA.pressing() || ctrPrimary.ButtonUp.pressing() || ctrPrimary.ButtonDown.pressing()){wait(20);}//wait for all buttons to be released
                    while(!(ctrPrimary.ButtonA.pressing() || ctrPrimary.ButtonUp.pressing() || ctrPrimary.ButtonDown.pressing())){//Waits for a button to be pressed to prevent controller lag
                        if (isField()){//If the robot is connected to the field, display message to remove the cable
                            ctrPrimary.Screen.clearScreen();
                            ctrPrimary.Screen.setCursor(1,0);
                            ctrPrimary.Screen.print("Remove Field Cable");
                            while (isField()){//Wait for field cable to be removed
                                wait(20);
                            }
                            break;//Break the loop to redisplay the options
                        }
                        wait(20);
```

```cpp
                        }
                    }
                }
};
bool confirmAuton(){
    if (mode == 1 && !ctrPrimary.ButtonUp.pressing()){
        return true;
    }
    if (mode == 2 && compControl.isAutonomous() && compControl.isEnabled() && isField()){
        return true;
    }
    return false;
}
bool confirmDriver(){
    if (mode == 0 && !ctrPrimary.ButtonUp.pressing()){
        return true;
    }
    if (mode == 2 && compControl.isDriverControl() && compControl.isEnabled() &&
    isField()){
        return true;
    }
    return false;
}
int selectAutonomous(){//method for selecting autons
    DisplaySelection selectAuton = DisplaySelection(5);//create display selection object
    strcpy(selectAuton.text[0], "Bypass");//place names of autons in array
    strcpy(selectAuton.text[1], "Skills");
    strcpy(selectAuton.text[2], "Game 6");
    strcpy(selectAuton.text[3], "Game 5");
    strcpy(selectAuton.text[4], "Game 1 :(");
    return selectAuton.select();
}
void colorSelect(){//method for selecting field color
    DisplaySelection selectColor = DisplaySelection(2);//create display object
    strcpy(selectColor.text[0], "Red");//set array values to colors
    strcpy(selectColor.text[1], "Blue");
    strcpy(selectColor.text[2], "");
    colorRed = (selectColor.select() == 0);
}
void displayLevels(){
    printf("Battery: %d\n", (int)Brain.Battery.capacity(percent));
    printf("Drive Left Front: %d\n", (int)mtrLeft.temperature(percent));
    printf("Drive Left Back: %d\n", (int)mtrLeftFront.temperature(percent));
    printf("Drive Right Front: %d\n", (int)mtrRight.temperature(percent));
    printf("Drive Right Back: %d\n", (int)mtrRightFront.temperature(percent));
    printf("Arm: %d\n", (int)mtrArm.temperature(percent));
    printf("Ramp: %d\n", (int)mtrRampLift.temperature(percent));
    printf("Intake Left: %d\n", (int)mtrIntakeLeft.temperature(percent));
    printf("Intake Right: %d\n\n", (int)mtrIntakeRight.temperature(percent));
}
int main() {
    ctrPrimary.Screen.clearScreen();
    ctrPrimary.Screen.setCursor(1,0);
    while(true){
        ctrPrimary.ButtonLeft.pressed(displayLevels);
        DisplaySelection selectMode = DisplaySelection(3); //Create Display object
        strcpy(selectMode.text[0], "Driver Control");//set values in array to options
        strcpy(selectMode.text[1], "Autonomous");
        strcpy(selectMode.text[2], "Field Control");
        strcpy(selectMode.text[3], "");
        mode = selectMode.select();
        //mode = 2;
        //colorRed = true;
        if (mode == 1 || mode == 2){
            calibrateGyros();
            autonMode = selectAutonomous();
            //autonMode = 2;
        }
        colorSelect();
        clearMotorRotations();
        if(mode == 0){//Runs driver control
            ctrPrimary.Screen.clearScreen();
            vex::task runDriver = vex::task(driver);
            while (confirmDriver()){wait(20);}
            runDriver.stop();
            stopAllMotors();
            while(ctrPrimary.ButtonUp.pressing()){wait(20);}//wait for exit button to be released
        }
        if (mode == 1){//Runs an auton
            ctrPrimary.Screen.clearScreen();
            vex::task runAuton = vex::task(auton);
            while (confirmAuton()){wait(20);}
            runAuton.stop();
            stopAllMotors();
            while(ctrPrimary.ButtonUp.pressing()){wait(20);}//wait for exit button to be released
        }
        if (mode == 2){

            while(true){//loop for competition
                if (!isField()){//Waits for the user to connect to the field after selections are made
                    ctrPrimary.Screen.clearScreen();
                    ctrPrimary.Screen.setCursor(1,0);
                    ctrPrimary.Screen.print("Connect to Field");
                    ctrPrimary.Screen.newLine();
                    ctrPrimary.Screen.print("(B) Close");
                    while(!ctrPrimary.ButtonB.pressing() && !isField()){wait(20);}
                    if(ctrPrimary.ButtonB.pressing()){
                        break;
                    }
                }
                while(!compControl.isEnabled()){//Wait while the robot is disabled
                    ctrPrimary.Screen.clearScreen();
                    ctrPrimary.Screen.setCursor(1,0);
                    ctrPrimary.Screen.print("Disabled");
                    while(!compControl.isEnabled()){wait(20);}
                    ctrPrimary.Screen.clearScreen();
                }

                if(compControl.isEnabled() && compControl.isAutonomous()){//runs auton when enabled and autonomous
                    vex::task runAuton(auton);
                    while (confirmAuton()){wait(20);}
                    runAuton.stop();
                }
                if(compControl.isEnabled() && compControl.isDriverControl()){//runs driver control when enabled and driver control
                    vex::task runDriver(driver);
                    while (confirmDriver()){wait(20);}
                    runDriver.stop();
                }
                stopAllMotors();
            }
            stopAllMotors();
        }
    }
}
```