Student Name: Program

Assignment:

Notes:

Project Name: HIStakesREDRIGHT

Project Type: C++

Date: Sun Dec 22 2024

```
#pragma region VEXcode Generated Robot Configuration
// Make sure all required headers are included.
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>


#include "vex.h"

using namespace vex;

// Brain should be defined by default
brain Brain;


// START V5 MACROS
#define waitUntil(condition)                                                   \
  do {                                                                         \
    wait(5, msec);                                                             \
  } while (!(condition))

#define repeat(iterations)                                                     \
  for (int iterator = 0; iterator < iterations; iterator++)
// END V5 MACROS


// Robot configuration code.
triport Expander = triport(PORT20);
// AI Classification Competition Element IDs
enum gameElements {
  mobileGoal,
  redRing,
  blueRing,
};

competition Competition;

controller Controller1 = controller(primary);
motor Left1 = motor(PORT1, ratio6_1, true);

motor Left2 = motor(PORT2, ratio6_1, false);

motor Left3 = motor(PORT3, ratio6_1, true);

motor Right1 = motor(PORT13, ratio6_1, false);

motor Right2 = motor(PORT16, ratio6_1, true);

motor Right3 = motor(PORT6, ratio6_1, false);

motor Intake_default = motor(PORT15, ratio6_1, false);

// AI Vision Color Descriptions
// AI Vision Code Descriptions
vex::aivision AIVision19(PORT19, aivision::ALL_AIOBJS);
```

```

inertial Inertial_Sensor = inertial(PORT10);

motor ArmThingMotorA = motor(PORT17, ratio18_1, false);
motor ArmThingMotorB = motor(PORT18, ratio18_1, true);
motor_group ArmThing = motor_group(ArmThingMotorA, ArmThingMotorB);

digital_out Mobile_Goal_Piston = digital_out(Expander.A);


// generating and setting random seed
void initializeRandomSeed(){
  int systemTime = Brain.Timer.systemHighResolution();
  double batteryCurrent = Brain.Battery.current();
  double batteryVoltage = Brain.Battery.voltage(voltageUnits::mV);

  // Combine these values into a single integer
  int seed = int(batteryVoltage + batteryCurrent * 100) + systemTime;

  // Set the seed
  srand(seed);
}



void vexcodeInit() {

  //Initializing random seed.
  initializeRandomSeed();
}


// Helper to make playing sounds from the V5 in VEXcode easier and
// keeps the code cleaner by making it clear what is happening.
void playVexcodeSound(const char *soundName) {
  printf("VEXPlaySound:%s\n", soundName);
  wait(5, msec);
}



// define variable for remote controller enable/disable
bool RemoteControlCodeEnabled = true;

#pragma endregion VEXcode Generated Robot Configuration

/*----------------------------------------------------------------------------*/
/*                                                                            */
/*    Module:       main.cpp                                                  */
/*    Author:       {author}                                                  */
/*    Created:      {date}                                                    */
/*    Description:  V5 project                                                */
/*                                                                            */
/*----------------------------------------------------------------------------*/

// Include the V5 Library
#include "vex.h"
```

```

// Allows for easier use of the VEX Library
using namespace vex;

event message1 = event();

int AIVision11_objectIndex = 0, Brain_precision = 0, Console_precision = 0, Controller1_precision = 0;

float myVariable, TempDriveTrain, MobileGoalPiston, TolerenceTime, X_axis, Y_axis, Error, InitialPlace, DistanceNeeded, Integral, prevError, Derivative;

bool test;

bool Intake_vari;

bool Intake_vari2;

bool AButton;

bool ArmFlipFlop;


motor_group RightSide(Right1, Right2, Right3);
motor_group LeftSide(Left1, Left2, Left3);

smartdrive Drive(LeftSide, RightSide, Inertial_Sensor, 260, 380, 385, mm, 1.67);

// "when started" hat block
int whenStarted1() {
  while (true) {

  }
  return 0;
}




//(distance * 360) / (wheel circumference * gear ratio)
//so (distance * 360.0) / ((PI * 3.25) * (36.0/60.0))
// 12 inches equals about 705 degrees needed
// "when autonomous" hat block
int onauton_autonomous_0() {
  InitialPlace = ((LeftSide.position(degrees) + RightSide.position(degrees)) / 2);
  DistanceNeeded = 1350 + InitialPlace;
  while (true) {
    Error = DistanceNeeded - ((LeftSide.position(degrees) + RightSide.position(degrees)) / 2);
    Integral = Integral + Error;
    if (Error == 0 or Error < 0) {
      Integral = 0;
    }
    if (Error > 1350 or Error < -50) {
      Integral = 0;
    }
    Derivative = Error - prevError;
    prevError = Error;
    RightSide.spin(forward, Error * 0 + Integral * 0 + Derivative * 0, volt);
    LeftSide.spin(forward, Error * 0 + Integral * 0 + Derivative * 0, volt);
    if (Error < 10 && Error > -10) {
      TolerenceTime = TolerenceTime + 15;
    } else {
        TolerenceTime = 0;
    }
   if (TolerenceTime == 45) {
      break;
    }
    wait(15, msec);
  }
  Mobile_Goal_Piston.set(true);
  Drive.turnToHeading(90, degrees);
  while (true) {

  }
  return 0;
}

// "when autonomous" hat block
int onauton_autonomous_1() {

  return 0;
}


// "when started" hat block
int whenStarted5() {
  Intake_default.setVelocity(600, rpm);
  Intake_vari = false;
  Intake_vari2 = false;
  while (true) {
    if (Intake_vari2 == false) {
      //AIVision19.takeSnapshot(blueRing);
      if (AIVision19.objectCount > 0) {
        Intake_default.spinFor(forward, 360.0, degrees);
        wait(0.5, seconds);
      }
    }
  }
  return 0;
}

// "when started" hat block
int whenStarted6() {
  while (true) {
    if (Controller1.ButtonR2.pressing()) {
      Intake_default.spin(reverse);
    }
      else if (!Controller1.ButtonR2.pressing()) {
        wait(1, msec);
        Intake_default.stop();
      }
    if (Controller1.ButtonR1.pressing()) {
      Intake_default.spin(forward);
    }
      else if (!Controller1.ButtonR1.pressing()) {
        wait(1, msec);
        Intake_default.stop();
      }
  }
  return 0;
}

// "when started" hat block
int whenStarted7() {
  LeftSide.spin(forward);
  RightSide.spin(forward);
  while (true) {
    LeftSide.setVelocity(Controller1.Axis3.position(), percent);
    RightSide.setVelocity(Controller1.Axis2.position(), percent);
    if (LeftSide.isDone()) {
      LeftSide.setStopping(brake);
    }
    if (RightSide.isDone()){
      RightSide.setStopping(brake);    }
  }
  return 0;
}

int whenStarted12() {
  while (true) {
    if (Controller1.ButtonA.pressing() && Intake_vari2 == false) {
      Intake_vari2 = true;
    }
      else if (Controller1.ButtonA.pressing() && Intake_vari2 == true) {
        Intake_vari2 = false;
      }
  }
  return 0;
}




// "when started" hat block
int whenStarted8() {

  return 0;
}

// "when started" hat block
int whenStarted9() {
  MobileGoalPiston = 0.0;
  while (true) {
    wait(0.35, seconds);
    if (Controller1.ButtonL2.pressing()) {
      MobileGoalPiston = MobileGoalPiston + 1.0;
    }
  wait(5, msec);
  }
  return 0;
```

```
}

// "when started" hat block
int whenStarted10() {
  ArmThing.setStopping(hold);
  ArmThing.setVelocity(100, percent);
  ArmThing.setMaxTorque(100, percent);
  while (true) {
    if (Controller1.ButtonL1.pressing()) {
      ArmThing.spin(forward);
    }
      else if (Controller1.ButtonA.pressing()) {
        ArmThing.spin(reverse);
      }
        else {
          ArmThing.setStopping(hold);
        }
  }
  return 0;
}

// "when started" hat block
int whenStarted11() {
  Mobile_Goal_Piston.set(false);
  while (true) {
    if (!(fmod(MobileGoalPiston,2.0) == 0.0)) {
      Mobile_Goal_Piston.set(true);
    } else if (fmod(MobileGoalPiston,2.0) == 0.0) {
      Mobile_Goal_Piston.set(false);
    } else {
    }
  wait(5, msec);
  }
  return 0;
}



// "when driver control" hat block
int ondriver_drivercontrol_0() {
  LeftSide.spin(forward);
  RightSide.spin(forward);
  while (true) {
    LeftSide.setVelocity(Controller1.Axis3.position(), percent);
    RightSide.setVelocity(Controller1.Axis2.position(), percent);
    if (LeftSide.isDone()) {
      LeftSide.setStopping(brake);
    }
    if (RightSide.isDone()){
      RightSide.setStopping(brake);    }
  }
  return 0;
}

int driver_control1() {
  while (true) {
    if (!(fmod(MobileGoalPiston,2.0) == 0.0)) {
      Mobile_Goal_Piston.set(true);
    } else if (fmod(MobileGoalPiston,2.0) == 0.0) {
      Mobile_Goal_Piston.set(false);
    } else {
    }
  wait(5, msec);
  }
}

int driver_control2() {
  while (true) {
    if (Controller1.ButtonR2.pressing()) {
      Intake_default.spin(reverse);
    }
      else if (!Controller1.ButtonR2.pressing()) {
        wait(1, msec);
        Intake_default.stop();
      }
    if (Controller1.ButtonR1.pressing()) {
      Intake_default.spin(forward);
    }
      else if (!Controller1.ButtonR1.pressing()) {
        wait(1, msec);
        Intake_default.stop();
      }
  }
}



void VEXcode_driver_task() {
//  // Start the driver control tasks....
  vex::task drive0(ondriver_drivercontrol_0);
  //vex::task drive1(driver_control1);
  //vex::task drive2(driver_control2);
  //vex::task drive3(driver_control3);
  while(Competition.isDriverControl() && Competition.isEnabled()) {this_thread::sleep_for(10);}
  drive0.stop();
  //drive1.stop();
  //drive2.stop();
  //drive3.stop();
  return;
}

void VEXcode_auton_task() {
  // Start the auton control tasks....
  vex::task auto0(onauton_autonomous_0);
//vex::task auto1(onauton_autonomous_1);
//vex::task auto2(onauton_autonomous_2);
  while(Competition.isAutonomous() && Competition.isEnabled()) {this_thread::sleep_for(10);}
  auto0.stop();
//auto1.stop();
//auto2.stop();
  return;
}
```

```cpp


int main() {
  vex::competition::bStopTasksBetweenModes = false;
  Competition.autonomous(VEXcode_auton_task);
  Competition.drivercontrol(VEXcode_driver_task);

  // Initializing Robot Configuration. DO NOT REMOVE!
  vexcodeInit();

  //vex::task ws1(whenStarted2);
  //vex::task ws2(whenStarted3);
  //vex::task ws3(whenStarted4);
  vex::task ws4(whenStarted5);
  vex::task ws5(whenStarted6);
  vex::task ws6(whenStarted7);
  //vex::task ws7(whenStarted8);
  vex::task ws8(whenStarted9);
  vex::task ws9(whenStarted10);
  vex::task ws10(whenStarted11);
  //vex::task ws11(whenStarted12);
  whenStarted1();
}
```