



Student Name: Program

Assignment:

Notes:

Project Name: HlStakesREDRIGHT

Project Type: C++

Date: Sun Dec 22 2024

```

1  #pragma region VEXcode Generated Robot Configuration
2  // Make sure all required headers are included.
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <stdbool.h>
6  #include <math.h>
7  #include <string.h>
8
9
10 #include "vex.h"
11
12 using namespace vex;
13
14 // Brain should be defined by default
15 brain Brain;
16
17
18 // START V5 MACROS
19 #define waitUntil(condition)
20     do {
21         wait(5, msec);
22     } while (!(condition))
23
24 #define repeat(iterations)
25     for (int iterator = 0; iterator < iterations; iterator++)
26 // END V5 MACROS
27
28
29 // Robot configuration code.
30 triport Expander = triport(PORT20);
31 // AI Classification Competition Element IDs
32 enum gameElements {
33     mobileGoal,
34     redRing,
35     blueRing,
36 };
37
38 competition Competition;
39
40 controller Controller1 = controller(primary);
41 motor Left1 = motor(PORT1, ratio6_1, true);
42
43 motor Left2 = motor(PORT2, ratio6_1, false);
44
45 motor Left3 = motor(PORT3, ratio6_1, true);
46
47 motor Right1 = motor(PORT13, ratio6_1, false);
48
49 motor Right2 = motor(PORT16, ratio6_1, true);
50
51 motor Right3 = motor(PORT6, ratio6_1, false);
52
53 motor Intake_default = motor(PORT15, ratio6_1, false);
54
55 // AI Vision Color Descriptions
56 // AI Vision Code Descriptions
57 vex::aivision AIVision19(PORT19, aivision::ALL_AIOBJS);

```

```

58
59  inertial Inertial_Sensor = inertial(PORT10);
60
61  motor ArmThingMotorA = motor(PORT17, ratio18_1, false);
62  motor ArmThingMotorB = motor(PORT18, ratio18_1, true);
63  motor_group ArmThing = motor_group(ArmThingMotorA, ArmThingMotorB);
64
65  digital_out Mobile_Goal_Piston = digital_out(Expander.A);
66
67
68  // generating and setting random seed
69  void initializeRandomSeed(){
70      int systemTime = Brain.Timer.systemHighResolution();
71      double batteryCurrent = Brain.Battery.current();
72      double batteryVoltage = Brain.Battery.voltage(voltageUnits::mV);
73
74      // Combine these values into a single integer
75      int seed = int(batteryVoltage + batteryCurrent * 100) + systemTime;
76
77      // Set the seed
78      srand(seed);
79  }
80
81
82
83  void vexcodeInit() {
84
85      //Initializing random seed.
86      initializeRandomSeed();
87  }
88
89
90  // Helper to make playing sounds from the V5 in VEXcode easier and
91  // keeps the code cleaner by making it clear what is happening.
92  void playVexcodeSound(const char *soundName) {
93      printf("VEXPlaySound:%s\n", soundName);
94      wait(5, msec);
95  }
96
97
98
99  // define variable for remote controller enable/disable
100  bool RemoteControlCodeEnabled = true;
101
102  #pragma endregion VEXcode Generated Robot Configuration
103
104  /*-----*/
105  /*
106  /*   Module:      main.cpp
107  /*   Author:      {author}
108  /*   Created:     {date}
109  /*   Description: V5 project
110  /*
111  /*-----*/
112
113  // Include the V5 Library
114  #include "vex.h"

```

```

115
116 // Allows for easier use of the VEX Library
117 using namespace vex;
118
119 event message1 = event();
120
121 int AIVision11_objectIndex = 0, Brain_precision = 0, Console_precision = 0, Contro
  ller1_precision = 0;
122
123 float myVariable, TempDriveTrain, MobileGoalPiston, TolarenceTime, X_axis, Y_axi
  s, Error, InitialPlace, DistanceNeeded, Integral, prevError, Derivative;
124
125 bool test;
126
127 bool Intake_vari;
128
129 bool Intake_vari2;
130
131 bool AButton;
132
133 bool ArmFlipFlop;
134
135
136 motor_group RightSide(Right1, Right2, Right3);
137 motor_group LeftSide(Left1, Left2, Left3);
138
139 smartdrive Drive(LeftSide, RightSide, Inertial_Sensor, 260, 380, 385, mm, 1.67);
140
141 // "when started" hat block
142 int whenStarted1() {
143     while (true) {
144
145     }
146     return 0;
147 }
148
149
150
151
152 //((distance * 360) / (wheel circumference * gear ratio)
153 //so (distance * 360.0) / ((PI * 3.25) * (36.0/60.0))
154 // 12 inches equals about 705 degrees needed
155 // "when autonomous" hat block
156 int onauton_autonomous_0() {
157     InitialPlace = ((LeftSide.position(degrees) + RightSide.position(degrees)) / 2);
158     DistanceNeeded = 1350 + InitialPlace;
159     while (true) {
160         Error = DistanceNeeded - ((LeftSide.position(degrees) + RightSide.position(deg
  rees)) / 2);
161         Integral = Integral + Error;
162         if (Error == 0 or Error < 0) {
163             Integral = 0;
164         }
165         if (Error > 1350 or Error < -50) {
166             Integral = 0;
167         }
168         Derivative = Error - prevError;

```

```

169     prevError = Error;
170     RightSide.spin(forward, Error * 0 + Integral * 0 + Derivative * 0, volt);
171     LeftSide.spin(forward, Error * 0 + Integral * 0 + Derivative * 0, volt);
172     if (Error < 10 && Error > -10) {
173         ToleranceTime = ToleranceTime + 15;
174     } else {
175         ToleranceTime = 0;
176     }
177     if (ToleranceTime == 45) {
178         break;
179     }
180     wait(15, msec);
181 }
182 Mobile_Goal_Piston.set(true);
183 Drive.turnToHeading(90, degrees);
184 while (true) {
185
186 }
187 return 0;
188 }
189
190 // "when autonomous" hat block
191 int onauton_autonomous_1() {
192
193     return 0;
194 }
195
196
197 // "when started" hat block
198 int whenStarted5() {
199     Intake_default.setVelocity(600, rpm);
200     Intake_vari = false;
201     Intake_vari2 = false;
202     while (true) {
203         if (Intake_vari2 == false) {
204             //AIVision19.takeSnapshot(blueRing);
205             if (AIVision19.objectCount > 0) {
206                 Intake_default.spinFor(forward, 360.0, degrees);
207                 wait(0.5, seconds);
208             }
209         }
210     }
211     return 0;
212 }
213
214 // "when started" hat block
215 int whenStarted6() {
216     while (true) {
217         if (Controller1.ButtonR2.pressing()) {
218             Intake_default.spin(reverse);
219         }
220         else if (!Controller1.ButtonR2.pressing()) {
221             wait(1, msec);
222             Intake_default.stop();
223         }
224         if (Controller1.ButtonR1.pressing()) {
225             Intake_default.spin(forward);

```

```

226     }
227     else if (!Controller1.ButtonR1.pressing()) {
228         wait(1, msec);
229         Intake_default.stop();
230     }
231 }
232 return 0;
233 }
234
235 // "when started" hat block
236 int whenStarted7() {
237     LeftSide.spin(forward);
238     RightSide.spin(forward);
239     while (true) {
240         LeftSide.setVelocity(Controller1.Axis3.position(), percent);
241         RightSide.setVelocity(Controller1.Axis2.position(), percent);
242         if (LeftSide.isDone()) {
243             LeftSide.setStopping(brake);
244         }
245         if (RightSide.isDone()){
246             RightSide.setStopping(brake);    }
247     }
248     return 0;
249 }
250
251 int whenStarted12() {
252     while (true) {
253         if (Controller1.ButtonA.pressing() && Intake_vari2 == false) {
254             Intake_vari2 = true;
255         }
256         else if (Controller1.ButtonA.pressing() && Intake_vari2 == true) {
257             Intake_vari2 = false;
258         }
259     }
260     return 0;
261 }
262
263
264
265
266 // "when started" hat block
267 int whenStarted8() {
268
269     return 0;
270 }
271
272 // "when started" hat block
273 int whenStarted9() {
274     MobileGoalPiston = 0.0;
275     while (true) {
276         wait(0.35, seconds);
277         if (Controller1.ButtonL2.pressing()) {
278             MobileGoalPiston = MobileGoalPiston + 1.0;
279         }
280     }
281     wait(5, msec);
282     return 0;

```

```

283     }
284
285 // "when started" hat block
286 int whenStarted10() {
287     ArmThing.setStopping(hold);
288     ArmThing.setVelocity(100, percent);
289     ArmThing.setMaxTorque(100, percent);
290     while (true) {
291         if (Controller1.ButtonL1.pressing()) {
292             ArmThing.spin(forward);
293         }
294         else if (Controller1.ButtonA.pressing()) {
295             ArmThing.spin(reverse);
296         }
297         else {
298             ArmThing.setStopping(hold);
299         }
300     }
301     return 0;
302 }
303
304 // "when started" hat block
305 int whenStarted11() {
306     Mobile_Goal_Piston.set(false);
307     while (true) {
308         if (!(fmod(MobileGoalPiston,2.0) == 0.0)) {
309             Mobile_Goal_Piston.set(true);
310         } else if (fmod(MobileGoalPiston,2.0) == 0.0) {
311             Mobile_Goal_Piston.set(false);
312         } else {
313         }
314         wait(5, msec);
315     }
316     return 0;
317 }
318
319
320
321 // "when driver control" hat block
322 int ondriver_drivercontrol_0() {
323     LeftSide.spin(forward);
324     RightSide.spin(forward);
325     while (true) {
326         LeftSide.setVelocity(Controller1.Axis3.position(), percent);
327         RightSide.setVelocity(Controller1.Axis2.position(), percent);
328         if (LeftSide.isDone()) {
329             LeftSide.setStopping(brake);
330         }
331         if (RightSide.isDone()){
332             RightSide.setStopping(brake);        }
333     }
334     return 0;
335 }
336
337 int driver_controll1() {
338     while (true) {
339         if (!(fmod(MobileGoalPiston,2.0) == 0.0)) {

```

```

340     Mobile_Goal_Piston.set(true);
341 } else if (fmod(MobileGoalPiston,2.0) == 0.0) {
342     Mobile_Goal_Piston.set(false);
343 } else {
344 }
345 wait(5, msec);
346 }
347 }
348
349 int driver_control2() {
350     while (true) {
351         if (Controller1.ButtonR2.pressing()) {
352             Intake_default.spin(reverse);
353         }
354         else if (!Controller1.ButtonR2.pressing()) {
355             wait(1, msec);
356             Intake_default.stop();
357         }
358         if (Controller1.ButtonR1.pressing()) {
359             Intake_default.spin(forward);
360         }
361         else if (!Controller1.ButtonR1.pressing()) {
362             wait(1, msec);
363             Intake_default.stop();
364         }
365     }
366 }
367
368
369
370 void VEXcode_driver_task() {
371     // // Start the driver control tasks....
372     vex::task drive0(ondriver_drivercontrol_0);
373     //vex::task drivel(driver_controll1);
374     //vex::task drive2(driver_control2);
375     //vex::task drive3(driver_control3);
376     while(Competition.isDriverControl() && Competition.isEnabled()) {this_thread::sl
    eep_for(10);}
377     drive0.stop();
378     //drivel.stop();
379     //drive2.stop();
380     //drive3.stop();
381     return;
382 }
383
384 void VEXcode_auton_task() {
385     // Start the auton control tasks....
386     vex::task auto0(onauton_autonomous_0);
387     //vex::task auto1(onauton_autonomous_1);
388     //vex::task auto2(onauton_autonomous_2);
389     while(Competition.isAutonomous() && Competition.isEnabled()) {this_thread::sleep
    _for(10);}
390     auto0.stop();
391     //auto1.stop();
392     //auto2.stop();
393     return;
394 }

```



```
396
397
398 int main() {
399     vex::competition::bStopTasksBetweenModes = false;
400     Competition.autonomous(VEXcode_auton_task);
401     Competition.drivercontrol(VEXcode_driver_task);
402
403     // Initializing Robot Configuration. DO NOT REMOVE!
404     vexcodeInit();
405
406     //vex::task ws1(whenStarted2);
407     //vex::task ws2(whenStarted3);
408     //vex::task ws3(whenStarted4);
409     vex::task ws4(whenStarted5);
410     vex::task ws5(whenStarted6);
411     vex::task ws6(whenStarted7);
412     //vex::task ws7(whenStarted8);
413     vex::task ws8(whenStarted9);
414     vex::task ws9(whenStarted10);
415     vex::task ws10(whenStarted11);
416     //vex::task ws11(whenStarted12);
417     whenStarted1();
418 }
```