```cpp
#include "main.h"
#include "EZ-Template/auton.hpp"
#include "EZ-Template/drive/drive.hpp"
#include "EZ-Template/util.hpp"
#include "autons.hpp"
#include "pros/adi.hpp"
#include "pros/misc.h"
#include "pros/motors.h"
#include "pros/motors.hpp"
#include "pros/rtos.hpp"



//BURNT OUT PORTS
//11

#define CATA_MOTOR_PORT 11
#define INTAKE_MOTOR_PORT 12

#define DRIVE_LB_PORT 16
#define DRIVE_LM_PORT 17
#define DRIVE_LF_PORT 19

#define DRIVE_RB_PORT 13
#define DRIVE_RM_PORT 14
#define DRIVE_RF_PORT 15

#define CATA_LIMIT_SWITCH_PORT 'G'

#define CATA_PRIME_BUTTON pros::E_CONTROLLER_DIGITAL_X
#define CATA_RE_PRIME_BUTTON pros::E_CONTROLLER_DIGITAL_A
#define CATA_LAUNCH_BUTTON pros::E_CONTROLLER_DIGITAL_LEFT
#define INTAKE_INTAKE_BUTTON pros::E_CONTROLLER_DIGITAL_L1
#define INTAKE_OUTTAKE_BUTTON pros::E_CONTROLLER_DIGITAL_L2
// #define CATA_SPIN_BUTTON pros::E_CONTROLLER_DIGITAL_UP
#define CATA_LAUNCH_LIMIT_BUTTON pros::E_CONTROLLER_DIGITAL_R1

#define AUTON_SELECT_BUTTON pros::E_CONTROLLER_DIGITAL_UP

#define EXPANSION_ACTIVATE_1 pros::E_CONTROLLER_DIGITAL_Y
#define EXPANSION_ACTIVATE_2 pros::E_CONTROLLER_DIGITAL_B
#define EXPANSION_ACTIVATE_3 pros::E_CONTROLLER_DIGITAL_RIGHT
#define EXPANSION_ACTIVATE_4 pros::E_CONTROLLER_DIGITAL_DOWN


pros::Controller controller (pros::E_CONTROLLER_MASTER);
pros::Motor Catapult(CATA_MOTOR_PORT, MOTOR_GEARSET_36, false);
pros::Motor Intake(INTAKE_MOTOR_PORT, MOTOR_GEARSET_18, false);

pros::ADIDigitalIn CataLimit('G');

pros::ADIDigitalOut ExpansionMech('B');

/////
// For instalattion, upgrading, documentations and tutorials, check out website!
// https://ez-robotics.github.io/EZ-Template/
/////

bool auton_finished = false;

int intake_spinning = 0;
int spin_cata_spinning = 0;
bool cata_moving = false;
bool cata_moving_limit = false;

int launch_cata_movement = 5400;
// 1480 silly
int prime_cata_movement = 1615; // old: 1550, 1810
int re_prime_cata_movement = 370;
int first_launch_cata_movement = launch_cata_movement + re_prime_cata_movement;
int intake_intake_velocity = 200; // old: 200
```

```
int intake_outtake_velocity = 200;

void expand() {
  ExpansionMech.set_value(true);
}

bool cata_limit_shoot = false;
bool cata_limit_prime = false;

void cata_limit_switch_task_function() {
  while (true) {
      if (CataLimit.get_value() == 1 && !cata_limit_shoot){
        Catapult.move_velocity(0);
      } else {
        Catapult.move_velocity(100);
        if (cata_limit_shoot) {
          pros::delay(750);
        }
        cata_limit_shoot = false;
      }
    }
    pros::delay(ez::util::DELAY_TIME);
  }

void shoot_cata(){
  cata_limit_shoot = true;
}

void test_cata() {
  // pros::delay(5000);
  // shoot_cata();
  return;
}

void spin_cata() {
    if (spin_cata_spinning == 1) {
      Catapult.move_velocity(100);
    } else if (intake_spinning == 2) {
      Catapult.move_velocity(-600);
    } else if (spin_cata_spinning == 0) {
      Catapult.move_velocity(0);
    } else {
    Catapult.move_velocity(0);
    return;
  }
}

void spin_intake() {
  if (intake_spinning == 1 && (CataLimit.get_value() == 1)) {
    Intake.move_velocity(intake_intake_velocity);
  } else if (intake_spinning == 2) {
    Intake.move_velocity(-intake_outtake_velocity);
  } else if (intake_spinning == 0) {
    Intake.move_velocity(0);
  } else {
    Intake.move_velocity(0);
    return;
  }
}

// // Chassis constructor
Drive chassis (
  // Left Chassis Ports (negative port will reverse it!)
  //   the first port is the sensored port (when trackers are not used!)
  {-DRIVE_LB_PORT, -DRIVE_LM_PORT, -DRIVE_LF_PORT}

  // Right Chassis Ports (negative port will reverse it!)
  //   the first port is the sensored port (when trackers are not used!)
  ,{DRIVE_RB_PORT, DRIVE_RM_PORT, DRIVE_RF_PORT}

  // IMU Port
  ,20
```

```
  // Wheel Diameter (Remember, 4" wheels are actually 4.125!)
  //    (or tracking wheel diameter)
  ,3.25

  // Cartridge RPM
  //   (or tick per rotation if using tracking wheels)
  ,600

  // External Gear Ratio (MUST BE DECIMAL)
  //    (or gear ratio of tracking wheel)
  // eg. if your drive is 84:36 where the 36t is powered, your RATIO would be 2.333.
  // eg. if your drive is 36:60 where the 60t is powered, your RATIO would be 0.6.
  ,1.666

  // Uncomment if using tracking wheels
  /*
  // Left Tracking Wheel Ports (negative port will reverse it!)
  // ,{1, 2} // 3 wire encoder
  // ,8 // Rotation sensor

  // Right Tracking Wheel Ports (negative port will reverse it!)
  // ,{-3, -4} // 3 wire encoder
  // ,-9 // Rotation sensor
  */

  // Uncomment if tracking wheels are plugged into a 3 wire expander
  // 3 Wire Port Expander Smart Port
  // ,1
);

/**
 * Runs initialization code. This occurs as soon as the program is started.
 *
 * All other competition modes are blocked by initialize; it is recommended
 * to keep execution time for this mode under a few seconds.
 */

const int DRIVE_SPEED = 110; // This is 110/127 (around 87% of max speed).  We don't suggest making this 127.
                             // If this is 127 and the robot tries to heading correct, it's only correcting by
                             // making one side slower.  When this is 87%, it's correcting by making one side
                             // faster and one side slower, giving better heading correction.

const int MID_DRIVE_SPEED = 95;
const int TURN_SPEED  = 90;
const int SWING_SPEED = 90;

 void default_constants() {
  chassis.set_slew_min_power(80, 80);
  chassis.set_slew_distance(7, 7);
  chassis.set_pid_constants(&chassis.headingPID, 11, 0, 20, 0);
  chassis.set_pid_constants(&chassis.forward_drivePID, 0.45, 0, 5, 0);
  chassis.set_pid_constants(&chassis.backward_drivePID, 0.45, 0, 5, 0);
  chassis.set_pid_constants(&chassis.turnPID, 5, 0.003, 35, 15);
  chassis.set_pid_constants(&chassis.swingPID, 7, 0, 45, 0);
}

void W_SKILLS() {
  // default_constants();

  // chassis.set_drive_pid(1.5, DRIVE_SPEED);
  // chassis.wait_drive();

  // Intake.move_relative(-600, intake_outtake_velocity);
  // pros::delay(500);

  // chassis.set_drive_pid(-15, DRIVE_SPEED);
  // chassis.wait_drive();

  // chassis.set_turn_pid(90, TURN_SPEED);
  // chassis.wait_drive();
```

```
// chassis.set_drive_pid(20, DRIVE_SPEED);
// chassis.wait_drive();

// Intake.move_relative(-600, intake_outtake_velocity);
// pros::delay(500);

// chassis.set_drive_pid(-12, DRIVE_SPEED);
// chassis.wait_drive();

// chassis.set_turn_pid(45, TURN_SPEED);
// chassis.wait_drive();

// expand();


default_constants();

chassis.set_drive_pid(1.5, DRIVE_SPEED);
chassis.wait_drive();

Intake.move_relative(-1200, intake_outtake_velocity);
pros::delay(1000);

chassis.set_drive_pid(-8, DRIVE_SPEED);
Intake.move_relative(200, intake_intake_velocity);
pros::delay(250);
chassis.wait_drive();

// Intake.move_relative(600, intake_intake_velocity);
// pros::delay(300);

// chassis.set_drive_pid(-4, DRIVE_SPEED);
// chassis.wait_drive();

chassis.set_turn_pid(50, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(-19, DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(-15, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(-5, DRIVE_SPEED);
chassis.wait_drive();

pros::delay(250);

shoot_cata();

pros::delay(1900);

Intake.move_relative(10000, intake_intake_velocity);

pros::delay(950);

shoot_cata();

Intake.move_velocity(0);

pros::delay(250);

chassis.set_turn_pid(-95, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(14, 120);
chassis.wait_drive();

pros::delay(500);
```

```
  Intake.move_relative(2200, 600);

  chassis.set_drive_pid(12, 40);
  chassis.wait_drive();

  chassis.set_drive_pid(-24, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-550, 400);

  chassis.set_turn_pid(-14.5, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-3, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(500);

  shoot_cata();

  pros::delay(1850);

  Intake.move_velocity(10000);

  pros::delay(1000);

  shoot_cata();

  pros::delay(250);

  Intake.move_velocity(0);

  chassis.set_drive_pid(-8, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(28, DRIVE_SPEED);
  chassis.wait_drive();

  expand();


  pros::delay(12500);

}

void skillz_auton() {
  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(4, DRIVE_SPEED);

  Intake.move_relative(-1500, intake_outtake_velocity);
  pros::delay(600);

  Intake.move_velocity(0);

  // chassis.wait_drive();

  // FIRST ROLLER

  // Intake.move_relative(-1300, intake_outtake_velocity);
  // pros::delay(800);

  // Intake.move_velocity(0);

  chassis.set_drive_pid(-13, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(90, TURN_SPEED);
```

```
chassis.wait_drive();

// chassis.set_drive_pid(-5.5, MID_DRIVE_SPEED);
// chassis.wait_drive();

// chassis.set_swing_pid(ez::RIGHT_SWING, 90, 60);
// chassis.wait_drive();

Intake.move_relative(10000, intake_intake_velocity);

chassis.set_drive_pid(28, 80);
chassis.wait_drive();

pros::delay(600);

Intake.move_velocity(0);

chassis.set_drive_pid(5, MID_DRIVE_SPEED);

pros::delay(300);

Intake.move_relative(-1300, intake_outtake_velocity);
pros::delay(800);

Intake.move_velocity(0);


// chassis.wait_drive();

// SECOND ROLLER

// Intake.move_relative(-1300, intake_outtake_velocity);
// pros::delay(800);


chassis.set_drive_pid(-8, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(0, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(-64, MID_DRIVE_SPEED, true);
chassis.wait_drive();

chassis.set_turn_pid(10, TURN_SPEED);
chassis.wait_drive();

pros::delay(200);

// FIRST SHOT

shoot_cata();

pros::delay(300);

chassis.set_turn_pid(-39, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(31, 70, true);
Intake.move_relative(10000, intake_intake_velocity);
chassis.wait_drive();

chassis.set_turn_pid(-135, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(35, 55, true);
chassis.wait_drive();

pros::delay(300);

chassis.set_turn_pid(-45, TURN_SPEED);
chassis.wait_drive();
```

```
chassis.set_drive_pid(-9, MID_DRIVE_SPEED);
chassis.wait_drive();

pros::delay(400);

Intake.move_velocity(0);

pros::delay(200);

// SECOND SHOT

shoot_cata();

pros::delay(200);

chassis.set_drive_pid(11, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(-135, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(40, 80);
chassis.wait_drive();

pros::delay(50);

Intake.move_relative(15000, intake_intake_velocity);

pros::delay(500);

chassis.set_drive_pid(22, 17);
chassis.wait_drive();

pros::delay(500);

// chassis.set_swing_pid(ez::RIGHT_SWING, -180, 50);

chassis.set_turn_pid(-180, TURN_SPEED);

pros::delay(300);
Intake.move_velocity(0);
chassis.wait_drive();



chassis.set_drive_pid(11, MID_DRIVE_SPEED);

pros::delay(500);

Intake.move_relative(-1300, intake_outtake_velocity);
pros::delay(800);

// chassis.wait_drive();

// THIRD ROLLER

// Intake.move_relative(-1300, intake_outtake_velocity);
// pros::delay(800);

chassis.set_drive_pid(-9, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(-85, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(-45, MID_DRIVE_SPEED, true);
chassis.wait_drive();

chassis.set_turn_pid(-95, TURN_SPEED);
chassis.wait_drive();
```

```
pros::delay(200);

// THIRD SHOT

shoot_cata();

pros::delay(200);

chassis.set_turn_pid(-65, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(46, MID_DRIVE_SPEED, true);
chassis.wait_drive();

Intake.move_relative(15000, intake_intake_velocity);

pros::delay(500);

chassis.set_drive_pid(19, 17);
chassis.wait_drive();

chassis.set_drive_pid(-1, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_swing_pid(ez::LEFT_SWING, -90, 60);
chassis.wait_drive();

Intake.move_velocity(0);

chassis.set_drive_pid(23, MID_DRIVE_SPEED);

pros::delay(600);

Intake.move_relative(-1300, intake_outtake_velocity);
pros::delay(800);

// chassis.wait_drive();

// FOURTH ROLLER

// Intake.move_relative(-1300, intake_outtake_velocity);
// pros::delay(800);

chassis.set_drive_pid(-9, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(-185, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(-45, MID_DRIVE_SPEED, true);
chassis.wait_drive();

chassis.set_turn_pid(-165, TURN_SPEED);
chassis.wait_drive();

pros::delay(200);

// FOURTH SHOT

shoot_cata();

pros::delay(200);

chassis.set_turn_pid(-220, TURN_SPEED);
chassis.wait_drive();

Intake.move_relative(12000, intake_intake_velocity);

chassis.set_drive_pid(32, 70);
chassis.wait_drive();

chassis.set_turn_pid(-315, TURN_SPEED);
```

```
  chassis.wait_drive();

  chassis.set_drive_pid(38, 55, true);
  chassis.wait_drive();

  chassis.set_turn_pid(-225, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-9, MID_DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_velocity(0);

  pros::delay(200);

  // FIFTH SHOT

  shoot_cata();

  pros::delay(200);

  chassis.set_drive_pid(11, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-315, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(38, 80);
  chassis.wait_drive();

  pros::delay(50);

  Intake.move_relative(15000, intake_intake_velocity);

  pros::delay(750);

  chassis.set_drive_pid(20, 17);
  chassis.wait_drive();

  chassis.set_turn_pid(-265, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-40, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-275, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(200);

  // SIXTH SHOT

  shoot_cata();

  pros::delay(200);

  chassis.set_turn_pid(-260, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(60, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-315, TURN_SPEED);
  chassis.wait_drive();

  expand();
}

void skills_auton() {
  // set default drivetrain constants
  default_constants();
  // drive into roller
```

```
chassis.set_drive_pid(1.5, DRIVE_SPEED);
chassis.wait_drive();

// FIRST ROLLER

Intake.move_relative(-1300, intake_outtake_velocity);
pros::delay(800);

Intake.move_velocity(0);

chassis.set_drive_pid(-4, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_swing_pid(ez::RIGHT_SWING, 90, 60);
chassis.wait_drive();

Intake.move_relative(10000, intake_intake_velocity);

chassis.set_drive_pid(33, 80);
chassis.wait_drive();

pros::delay(400);

Intake.move_velocity(0);

chassis.set_drive_pid(6, MID_DRIVE_SPEED);
chassis.wait_drive();


Intake.move_relative(-1300, intake_outtake_velocity);
pros::delay(800);

// chassis.set_turn_pid(0, TURN_SPEED);
// chassis.wait_drive();

chassis.set_drive_pid(-72, MID_DRIVE_SPEED, true);
chassis.wait_drive();

chassis.set_turn_pid(94, TURN_SPEED);
chassis.wait_drive();

pros::delay(100);

shoot_cata();

pros::delay(250);

chassis.set_turn_pid(126, TURN_SPEED);
chassis.wait_drive();

chassis.set_drive_pid(24, 80);
chassis.wait_drive();

pros::delay(200);

chassis.set_drive_pid(-4, MID_DRIVE_SPEED);
chassis.wait_drive();

Intake.move_relative(10000, intake_intake_velocity);

pros::delay(500);

chassis.set_drive_pid(23, 30);
chassis.wait_drive();

chassis.set_drive_pid(-42, MID_DRIVE_SPEED);
chassis.wait_drive();

chassis.set_turn_pid(94, TURN_SPEED);
chassis.wait_drive();

pros::delay(500);
```

```
  Intake.move_velocity(0);

  pros::delay(200);

  shoot_cata();
}

void left_two_disc_auton() {

  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  // drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-16.5, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(7000);

  chassis.set_drive_pid(-44, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(-35, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-2.5, MID_DRIVE_SPEED);
  chassis.wait_drive();

  // chassis.set_drive_pid(8, MID_DRIVE_SPEED);
  // chassis.wait_drive();

  // chassis.set_drive_pid(-9, 127);
  // chassis.wait_until(-3);
  //pros::delay(50);
  shoot_cata();
  // chassis.wait_drive();

  pros::delay(12000);

  // Intake.move_relative(10000, intake_intake_velocity);

  // pros::delay(1200);

  // shoot_cata();

  // Intake.move_velocity(0);

  // pros::delay(5000);
}

void left_awp_auton_1() {

  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  // drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-18.5, DRIVE_SPEED, true);
  chassis.wait_drive();
```

```
  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-44, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(-35, TURN_SPEED);
  chassis.wait_drive();

  // chassis.set_drive_pid(3, DRIVE_SPEED);
  // chassis.wait_drive();

  pros::delay(200);

  //pros::delay(50);
  shoot_cata();
  // chassis.wait_drive();

  pros::delay(400);


  Intake.move_relative(10000, intake_intake_velocity);

  chassis.set_swing_pid(ez::RIGHT_SWING, -135, 120);
  chassis.wait_drive();

  chassis.set_drive_pid(7, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(300);

  chassis.set_turn_pid(-48, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(300);

  Intake.move_velocity(0);

  chassis.set_drive_pid(-10, DRIVE_SPEED);
  pros::delay(225);
  shoot_cata();
  chassis.wait_drive();

  pros::delay(10000);

}

void left_two_disc_auton_part_two() {

  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  //;p0 drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-16.5, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  // // pros::delay(7000);

  // chassis.set_drive_pid(-44, DRIVE_SPEED, true);
  // chassis.wait_drive();

  // chassis.set_turn_pid(-35, TURN_SPEED);
  // chassis.wait_drive();
```

```
  // chassis.set_drive_pid(-3.5, MID_DRIVE_SPEED);
  // chassis.wait_drive();

  // // chassis.set_drive_pid(8, MID_DRIVE_SPEED);
  // // chassis.wait_drive();

  // // chassis.set_drive_pid(-9, 127);
  // // chassis.wait_until(-3);
  // //pros::delay(50);
  // shoot_cata();
  // // chassis.wait_drive();

  // pros::delay(12000);

  // Intake.move_relative(10000, intake_intake_velocity);

  // pros::delay(1200);

  // shoot_cata();

  // Intake.move_velocity(0);

  // pros::delay(5000);
}

void left_full_awp_four_disc_auton() {

  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  // drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-18.25, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-44, DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(-39, 60);
  chassis.wait_drive();

  // chassis.set_drive_pid(3, DRIVE_SPEED);
  // chassis.wait_drive();

  pros::delay(200);

  //pros::delay(50);
  shoot_cata();
  // chassis.wait_drive();

  pros::delay(400);


  Intake.move_relative(10000, intake_intake_velocity);

  chassis.set_swing_pid(ez::RIGHT_SWING, -135, 120);
  chassis.wait_drive();

  chassis.set_drive_pid(7, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(300);

  chassis.set_turn_pid(-48, TURN_SPEED);
  chassis.wait_drive();
```

```
  pros::delay(300);

  Intake.move_velocity(0);

  chassis.set_drive_pid(-10, DRIVE_SPEED);
  pros::delay(225);
  shoot_cata();
  chassis.wait_drive();

  chassis.set_swing_pid(ez::RIGHT_SWING, -135, 60);
  pros::delay(150);
  Intake.move_relative(10000, intake_intake_velocity);
  chassis.wait_drive();

  chassis.set_drive_pid(8, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_swing_pid(ez::RIGHT_SWING, -50, 90);
  chassis.wait_drive();

  chassis.set_drive_pid(-8, 120);
  chassis.wait_until(-4);


  Intake.move_velocity(0);

  shoot_cata();
  chassis.wait_drive();

  pros::delay(250);

  chassis.set_drive_pid(4, DRIVE_SPEED);
  chassis.wait_drive();

  // chassis.set_turn_pid(-135, TURN_SPEED);
  // chassis.wait_drive();

  chassis.set_swing_pid(ez::RIGHT_SWING, -135, 90);
  chassis.wait_drive();

  Intake.move_relative(10000, intake_intake_velocity);

  chassis.set_drive_pid(37, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_swing_pid(ez::LEFT_SWING, -85, 110);
  chassis.wait_drive();

  Intake.move_velocity(0);

  chassis.set_drive_pid(6.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600); // spin roller
  pros::delay(500);
  chassis.set_drive_pid(-2.5, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(5000);

}

void left_crazy_6_disc_auton() {
  default_constants();
  // drive into roller
  chassis.set_drive_pid(-23, 125);
  pros::delay(300);
  shoot_cata();
  chassis.wait_drive();

  chassis.set_drive_pid(8, DRIVE_SPEED);
```

```
  chassis.wait_drive();

}

void left_full_awp_auton() {
  /* one disc in catapult, one high up in intake */

  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  // drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-8, DRIVE_SPEED);
  Intake.move_relative(200, intake_intake_velocity);
  pros::delay(250);
  chassis.wait_drive();
  // move to AWP line and line up for shot
  chassis.set_turn_pid(50, TURN_SPEED);
  chassis.wait_drive();
  chassis.set_drive_pid(-19, DRIVE_SPEED);
  chassis.wait_drive();
  chassis.set_turn_pid(-15, TURN_SPEED);
  chassis.wait_drive();
  chassis.set_drive_pid(-5.5, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(250); // settle before shooting
  shoot_cata();
  pros::delay(1900); // wait for catapult to come back down
  // intake second disc
  Intake.move_relative(10000, intake_intake_velocity);
  pros::delay(950);
  shoot_cata();
  Intake.move_velocity(0); // stop intake
  pros::delay(250);

  // move to center of field
  // chassis.set_drive_pid(2, DRIVE_SPEED);
  // chassis.wait_drive();
  chassis.set_turn_pid(46, MID_DRIVE_SPEED);
  chassis.wait_drive();
  chassis.set_drive_pid(-34, MID_DRIVE_SPEED);
  chassis.wait_drive();
  chassis.set_turn_pid(-95, MID_DRIVE_SPEED);
  chassis.wait_drive();

  // intake the three discs in a line
  Intake.move_relative(100000, intake_intake_velocity);
  chassis.set_drive_pid(12, 70);
  chassis.wait_drive();
  chassis.set_turn_pid(-135, TURN_SPEED);
  chassis.wait_drive();
  chassis.set_drive_pid(73, MID_DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_velocity(0); // stop intake

  // turn and move into roller
  chassis.set_turn_pid(-90, TURN_SPEED);
  chassis.wait_drive();
  chassis.set_drive_pid(4.5, MID_DRIVE_SPEED);
  chassis.wait_drive();
  Intake.move_relative(-600, 600); // spin roller
  pros::delay(500);
  chassis.set_drive_pid(-2.5, DRIVE_SPEED);
  chassis.wait_drive();
  pros::delay(10000);// wait for end of autonomous
}
```

```
void left_awp_auton() {

  default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);

  chassis.set_drive_pid(-8, DRIVE_SPEED);
  Intake.move_relative(200, intake_intake_velocity);
  pros::delay(250);
  chassis.wait_drive();

  // Intake.move_relative(600, intake_intake_velocity);
  // pros::delay(300);

  // chassis.set_drive_pid(-4, DRIVE_SPEED);
  // chassis.wait_drive();

  chassis.set_turn_pid(50, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-19, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-15, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-5, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(250);

  shoot_cata();

  pros::delay(1900);

  Intake.move_relative(10000, intake_intake_velocity);

  pros::delay(950);

  shoot_cata();

  Intake.move_velocity(0);

  // pros::delay(250);

  // chassis.set_turn_pid(-95, MID_DRIVE_SPEED);
  // chassis.wait_drive();

  // chassis.set_drive_pid(14, 120);
  // chassis.wait_drive();

  // pros::delay(500);



  // Intake.move_relative(2200, 600);

  // chassis.set_drive_pid(12, 40);
  // chassis.wait_drive();

  // chassis.set_drive_pid(-24, DRIVE_SPEED);
  // chassis.wait_drive();

  // Intake.move_relative(-550, 400);

  // chassis.set_turn_pid(-14.5, TURN_SPEED);
  // chassis.wait_drive();
```

```
  // chassis.set_drive_pid(-3, DRIVE_SPEED);
  // chassis.wait_drive();

  // pros::delay(500);

  // shoot_cata();

  // pros::delay(1850);

  // Intake.move_velocity(10000);

  // pros::delay(1000);

  // shoot_cata();

  // Intake.move_velocity(0);


  pros::delay(12500);
}

void right_awp_auton() {

}

void left_roller_auton() {

}

void right_roller_auton() {

}

void left_elim_auton() {
  // set default drivetrain constants
  default_constants();
  // drive into roller
  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();
  // set intake to move -600 ticks (enough to move the roller 1/4 rotation)
  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);
  // drive backward from roller, and reintake the disc
  chassis.set_drive_pid(-6, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-32, MID_DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(-26, TURN_SPEED);
  chassis.wait_drive();

  // chassis.set_drive_pid(-3, 120);
  // chassis.wait_until(-0.75);
  //pros::delay(50);

  chassis.set_drive_pid(-10, MID_DRIVE_SPEED, true);
  chassis.wait_drive();

  shoot_cata();
  chassis.wait_drive();

  pros::delay(1400);

  Intake.move_relative(10000, intake_intake_velocity);

  pros::delay(1300);
```

```
  shoot_cata();

  pros::delay(200);

  chassis.set_turn_pid(-110, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(400);

  chassis.set_drive_pid(5.5, 25);
  chassis.wait_drive();

  chassis.set_turn_pid(-32, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(900);

  shoot_cata();

  pros::delay(5000);

}

void right_elim_auton() {
  default_constants();

  // chassis.set_turn_pid(15, TURN_SPEED);
  // chassis.wait_drive();

  // chassis.set_drive_pid(-38, 125);
  // // chassis.wait_drive();

  // chassis.wait_until(-31);

  // shoot_cata();

  // chassis.wait_drive();

  // pros::delay(10000);

  chassis.set_drive_pid(-48, MID_DRIVE_SPEED, true);
  chassis.wait_drive();

  chassis.set_turn_pid(24, TURN_SPEED);
  chassis.wait_drive();

  shoot_cata();

  pros::delay(1200);

  Intake.move_relative(10000, intake_intake_velocity);

  pros::delay(800);

  shoot_cata();

  pros::delay(200);

  Intake.move_velocity(0);

  chassis.set_turn_pid(-17, TURN_SPEED);
  chassis.wait_drive();


}

void far_elim() {
  chassis.set_drive_pid(-38, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(38, TURN_SPEED);
  chassis.wait_drive();
```

```
  // first_launch_cata();
  shoot_cata();
  pros::delay(2000);

  Intake.move_relative(3000, intake_intake_velocity);

  pros::delay(750);

  Intake.move_velocity(0);

  // launch_cata();
  shoot_cata();

  pros::delay(250);

  chassis.set_drive_pid(4, 40);
  chassis.wait_drive();

  chassis.set_turn_pid(-45, TURN_SPEED);
  chassis.wait_drive();

  Intake.move_relative(10000, intake_intake_velocity);

  chassis.set_drive_pid(20, 40);
  chassis.wait_drive();

  chassis.set_turn_pid(28, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-6, 40);
  chassis.wait_drive();

  pros::delay(750);

  // launch_cata();
  shoot_cata();

  pros::delay(250);

  chassis.set_turn_pid(-45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(33, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_swing_pid(ez::LEFT_SWING, 0, 90);
  chassis.wait_drive();

  chassis.set_drive_pid(5, MID_DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-1.5, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(10000);

}

void drive_example() {
  // The first parameter is target inches
  // The second parameter is max speed the robot will drive at
  // The third parameter is a boolean (true or false) for enabling/disabling a slew at the start of drive motions
  // for slew, only enable it when the drive distance is greater then the slew distance + a few inches

  printf("drive example");

  default_constants();
```

```
  chassis.set_drive_pid(24, DRIVE_SPEED);
  chassis.wait_drive();
}

void test_fire() {
  // first_launch_cata();
  shoot_cata();
  pros::delay(250);
}

void near_roller() {
   default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-6, DRIVE_SPEED);
  chassis.wait_drive();
}

void near_full_AWP() {
  // The first parameter is target inches
  // The second parameter is max speed the robot will drive at
  // The third parameter is a boolean (true or false) for enabling/disabling a slew at the start of drive motions
  // for slew, only enable it when the drive distance is greater then the slew distance + a few inches

  printf("near awp");

  default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-9, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-66, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-42, TURN_SPEED);
  chassis.wait_drive();
  pros::delay(300);

  // launch_cata();
  // first_launch_cata();
  shoot_cata();
  pros::delay(500);

  chassis.set_turn_pid(-132, TURN_SPEED);
  chassis.wait_drive();

  pros::delay(500);
  Intake.move_relative(6000, 600);

  chassis.set_drive_pid(66, MID_DRIVE_SPEED);
  chassis.wait_drive();

  // chassis.set_turn_pid(-90, TURN_SPEED);
  chassis.set_swing_pid(ez::LEFT_SWING, -90, 90);
  chassis.wait_drive();

  chassis.set_drive_pid(10, MID_DRIVE_SPEED);
  chassis.wait_drive();
```

```
  Intake.move_velocity(0);

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-2, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_drive_brake(pros::E_MOTOR_BRAKE_COAST);

  pros::delay(13000);

}

void near_AWP() {
  // The first parameter is target inches
  // The second parameter is max speed the robot will drive at
  // The third parameter is a boolean (true or false) for enabling/disabling a slew at the start of drive motions
  // for slew, only enable it when the drive distance is greater then the slew distance + a few inches

  printf("near awp");

  default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-9, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-67, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-42, TURN_SPEED);
  chassis.wait_drive();
  pros::delay(300);

  // launch_cata();
  // first_launch_cata();
  shoot_cata();
  pros::delay(10000);

}


void near_four_disc() {
  printf("near four discs");

  default_constants();

  chassis.set_drive_pid(1.75, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, 600);
  pros::delay(500);

  chassis.set_drive_pid(-9, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(45, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-56, MID_DRIVE_SPEED);
  chassis.wait_drive();
```

```
  chassis.set_turn_pid(-35, TURN_SPEED);
  chassis.wait_drive();
  pros::delay(150);

  chassis.set_drive_pid(-4, MID_DRIVE_SPEED);
  chassis.wait_drive();
  pros::delay(300);
  // launch_cata();
  // first_launch_cata();
  shoot_cata();
  pros::delay(500);

  Intake.move_relative(10000, intake_intake_velocity);

  chassis.set_drive_pid(2, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-115, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(12, 40);
  chassis.wait_drive();

  chassis.set_turn_pid(-135, 40);
  chassis.wait_drive();

  chassis.set_drive_pid(20, 40);
  chassis.wait_drive();

  chassis.set_drive_pid(-23, MID_DRIVE_SPEED);
  Intake.move_velocity(0);
  chassis.wait_drive();

  pros::delay(300);
  // launch_cata();
  shoot_cata();

}

void near_four_disc_all_close() {
  printf("near four discs all close");

  default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-900, 600);
  pros::delay(1000);

  Intake.move_velocity(0);

  // chassis.set_drive_pid(-14, MID_DRIVE_SPEED);
  // chassis.wait_drive();

  chassis.set_swing_pid(ez::RIGHT_SWING, 45, -MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_swing_pid(ez::LEFT_SWING, -11, -MID_DRIVE_SPEED);
  chassis.wait_drive();

  // chassis.set_turn_pid(-11, MID_DRIVE_SPEED);
  // chassis.wait_drive();

  // first_launch_cata();
  shoot_cata();
  pros::delay(2250);

  Intake.move_relative(6000, intake_intake_velocity - 150);
  pros::delay(900);

  // launch_cata();
```

```
  shoot_cata();

  pros::delay(500);

  chassis.set_turn_pid(100, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(7, 60);
  chassis.wait_drive();

  chassis.set_drive_pid(-7, 60);
  chassis.wait_drive();

  chassis.set_turn_pid(-10, MID_DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-2, 60);
  chassis.wait_drive();

  pros::delay(250);

  // launch_cata();
  shoot_cata();

  pros::delay(10000);
}

void test_auton_selector_brain_task_function() {
  pros::lcd::register_btn0_cb(ez::as::page_down);
  pros::lcd::register_btn2_cb(ez::as::page_up);
}

void left_elim_auton_part_two() {
  default_constants();

  chassis.set_drive_pid(-28, DRIVE_SPEED);
  chassis.wait_drive();

  chassis.set_turn_pid(-38, TURN_SPEED);
  chassis.wait_drive();

  chassis.set_drive_pid(-19, DRIVE_SPEED);
  chassis.wait_drive();

  pros::delay(300);

  shoot_cata();

  pros::delay(200);




  chassis.set_swing_pid(ez::LEFT_SWING, 45,100);
  Intake.move_relative(10000, intake_intake_velocity);
  chassis.wait_drive();

  Intake.move_velocity(0);
}

void roller() {
  default_constants();

  chassis.set_drive_pid(1.5, DRIVE_SPEED);
  chassis.wait_drive();

  Intake.move_relative(-600, intake_outtake_velocity);
  pros::delay(500);

  chassis.set_drive_pid(-4, DRIVE_SPEED);
  chassis.wait_drive();
```

```
  pros::delay(10000);
}

void initialize() {
  // Print our branding over your terminal :D
  ez::print_ez_template();

  pros::delay(500); // Stop the user from doing anything while legacy ports configure.

  // Configure your chassis controls
  // chassis.toggle_modify_curve_with_controller(true); // Enables modifying the controller curve with buttons on the joysticks
  // chassis.set_active_brake(0.1); // Sets the active brake kP. We recommend 0.1.
  // chassis.set_curve_default(0, 0); // Defaults for curve. If using tank, only the first parameter is used. (Comment this line out if
you have an SD card!)
  default_constants(); // Set the drive to your own constants from autons.cpp!
  exit_condition_defaults(); // Set the exit conditions to your own constants from autons.cpp!

  Catapult.set_brake_mode(pros::E_MOTOR_BRAKE_HOLD);
  chassis.set_drive_brake(pros::E_MOTOR_BRAKE_COAST);
  chassis.set_curve_default(3.0,0);
  chassis.toggle_modify_curve_with_controller(false);

  // These are already defaulted to these buttons, but you can change the left/right curve buttons here!
  // chassis.set_left_curve_buttons (pros::E_CONTROLLER_DIGITAL_LEFT, pros::E_CONTROLLER_DIGITAL_RIGHT); // If using tank, only the left
side is used.
  // chassis.set_right_curve_buttons(pros::E_CONTROLLER_DIGITAL_Y,    pros::E_CONTROLLER_DIGITAL_A);

  // Autonomous Selector using LLEMU
  ez::as::auton_selector.add_autons({
    Auton("roller ez", roller),
    Auton("Left Two Disc Part 2", left_two_disc_auton_part_two),
    Auton("Left Elims part two", left_elim_auton_part_two),
    Auton("Four Disc AWP", left_full_awp_four_disc_auton),
    Auton("Left Elims", left_elim_auton),
    Auton("Right Elims", right_elim_auton),
    Auton("Skillz", skillz_auton),
    Auton("INSANE SKILLS WORLD RECORD", W_SKILLS),
    Auton("Left Side Two Disc", left_two_disc_auton),
    Auton("Left 6 Disc", left_crazy_6_disc_auton),
    Auton("test catapultie", test_cata),
    Auton("Left Full AWP", left_full_awp_auton),
    Auton("Left Only AWP", left_awp_auton),
    Auton("Far Elims", far_elim),
    Auton("Near Full AWP.", near_full_AWP),
    Auton("Near Four Disc.", near_four_disc),
    Auton("Near Four Disc All Close.", near_four_disc_all_close),
    Auton("Test Drive Forward", drive_example),
    Auton("Near AWP.", near_AWP),
    Auton("Test Fire", test_fire),
    Auton("Near Roller", near_roller)
    // Auton("Example Turn\n\nTurn 3 times.", turn_example),
    // Auton("Drive and Turn\n\nDrive forward, turn, come back. ", drive_and_turn),
    // Auton("Drive and Turn\n\nSlow down during drive.", wait_until_change_speed),
    // Auton("Swing Example\n\nSwing, drive, swing.", swing_example),
    // oAuton("Combine all 3 movements", combining_movements),
    // Auton("Interference\n\nAfter driving forward, robot performs differently if interfered or not.", interfered_example);
  });


  // // Initialize chassis and auton selector
  chassis.initialize();
  ez::as::initialize();

  pros::lcd::register_btn0_cb(ez::as::page_down);
  pros::lcd::register_btn2_cb(ez::as::page_up);

  // pros::delay(250);

  // while (! controller.get_digital(AUTON_SELECT_BUTTON)) {
  //   pros::delay(10);
  // }
```

```
  // pros::Task cata_limit_switch_task(cata_limit_switch_task_function);
  // pros::Task test_auton_selector_brain_task(test_auton_selector_brain_task_function);


  // pros::delay(500);
}



/**
 * Runs while the robot is in the disabled state of Field Management System or
 * the VEX Competition Switch, following either autonomous or opcontrol. When
 * the robot is enabled, this task will exit.
 */
void disabled() {
  // . . .
}



/**
 * Runs after initialize(), and before autonomous when connected to the Field
 * Management System or the VEX Competition Switch. This is intended for
 * competition-specific initialization routines, such as an autonomous selector
 * on the LCD.
 *
 * This task will exit when the robot is enabled and autonomous or opcontrol
 * starts.
 */
void competition_initialize() {
  // . . .
}



/**
 * Runs the user autonomous code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the autonomous
 * mode. Alternatively, this function may be called in initialize or opcontrol
 * for non-competition testing purposes.
 *
 * If the robot is disabled or communications is lost, the autonomous task
 * will be stopped. Re-enabling the robot will restart the task, not re-start it
 * from where it left off.
 */
void autonomous() {
  // chassis.reset_pid_targets(); // Resets PID targets to 0
  // chassis.reset_gyro(); // Reset gyro position to 0
  // chassis.reset_drive_sensor(); // Reset drive sensors to 0
  // chassis.set_drive_brake(MOTOR_BRAKE_HOLD); // Set motors to hold.  This helps autonomous consistency.
  chassis.reset_drive_sensor();
  chassis.reset_gyro();
  auton_finished = true;
  ez::as::auton_selector.call_selected_auton(); // Calls selected auton from autonomous selector.
}



/**
 * Runs the operator control code. This function will be started in its own task
 * with the default priority and stack size whenever the robot is enabled via
 * the Field Management System or the VEX Competition Switch in the operator
 * control mode.
 *
 * If no competition control is connected, this function will run immediately
 * following initialize().
 *
 * If the robot is disabled or communications is lost, the
 * operator control task will be stopped. Re-enabling the robot will restart the
 * task, not resume it from where it left off.
 */
```

```
bool launch_pressed = false;
bool launch_pressed_last = false;

bool launch_limit_pressed = false;
bool launch_limit_pressed_last = false;

bool prime_pressed = false;
bool prime_pressed_last = false;

bool re_prime_pressed = false;
bool re_prime_pressed_last = false;

bool intake_pressed = false;
bool intake_pressed_last = false;

bool outtake_pressed = false;
bool outtake_pressed_last = false;

bool spin_cata_pressed = false;
bool spin_cata_pressed_last = false;

void opcontrol() {
  // printf("something, anything at all.");
  // This is preference to what you like to drive on.
  // chassis.set_drive_brake(MOTOR_BRAKE_COAST);
  chassis.reset_drive_sensors_opcontrol();
  if (auton_finished) {
    auton_finished = false;
  }

  chassis.set_drive_brake(pros::E_MOTOR_BRAKE_COAST);

  // if (! auton_finished) {
    while (! controller.get_digital(AUTON_SELECT_BUTTON)) {
      pros::delay(10);
    }
  // }

  pros::Task cata_limit_switch_task(cata_limit_switch_task_function);

  while (true) {

    // chassis.tank(); // Tank control
    // chassis.arcade_standard(ez::SPLIT); // Standard split arcade
    // chassis.arcade_standard(ez::SINGLE); // Standard single arcade
    chassis.arcade_flipped(ez::SPLIT); // Flipped split arcade
    // chassis.arcade_flipped(ez::SINGLE); // Flipped single arcade

    if (controller.get_digital(CATA_LAUNCH_LIMIT_BUTTON)) {
      launch_limit_pressed = true;
    } else {
      launch_limit_pressed = false;
    }
    if (launch_limit_pressed && ! launch_limit_pressed_last) {
      shoot_cata();
    }
    launch_limit_pressed_last = launch_limit_pressed;

    if (Catapult.get_position()) {
      cata_moving = false;
    }

    if (controller.get_digital(INTAKE_INTAKE_BUTTON)) {
      intake_pressed = true;
    } else {
      intake_pressed = false;
    }

    if (controller.get_digital(INTAKE_OUTTAKE_BUTTON)) {
      outtake_pressed = true;
    } else {
      outtake_pressed = false;
```

```
    }

    if (intake_pressed) {
      intake_spinning = 1;
    } else if (outtake_pressed) {
      intake_spinning = 2;
    } else {
      intake_spinning = 0;
    }

    if (controller.get_digital(EXPANSION_ACTIVATE_1) &&
        controller.get_digital(EXPANSION_ACTIVATE_2) &&
        controller.get_digital(EXPANSION_ACTIVATE_3) &&
        controller.get_digital(EXPANSION_ACTIVATE_4)) {
          expand();
      }

    // if (controller.get_digital(CATA_SPIN_BUTTON)) {
    //   spin_cata_pressed = true;
    // } else {
    //   spin_cata_pressed = false;
    // }
    // if (spin_cata_pressed && ! spin_cata_pressed_last) {
    //   if (spin_cata_spinning == 0 || spin_cata_spinning == 1) {
    //     spin_cata_spinning = 1;
    //   } else if (spin_cata_spinning == 1) {
    //     spin_cata_spinning = 0;
    //     // spin_cata();
    //   }
    // }
    // spin_cata_pressed_last = spin_cata_pressed;

    spin_intake();

    pros::delay(ez::util::DELAY_TIME); // This is used for timer calculations!  Keep this ez::util::DELAY_TIME
  }
}
```