```
#include <Servo.h>  // standard servo library, probably not needed but here
just in case

// pin number of on-board LED
int ledPin = 13;

int debuggingMode = 0;        // go into debugging/testing mode to check if
seperate modules work
int PWMDebugPin = 11;
int PWMDebugTime = 500;
bool LightDebugStatus = true;

// test if the Arduino is alive
void WaitAndBlink( unsigned long DeltaMilliSec) {
  // wait DeltaMilliSec milliseconds, LED blinks as a sign of life
  // as time passes frequency increases
  unsigned long DeltaT = 0;
  unsigned long TZero = millis(); //get start time
  while (DeltaT < DeltaMilliSec) {
    unsigned long TCurrent = millis();
    DeltaT = TCurrent - TZero;    //compute elapsed time
    delay(500 - 400 * DeltaT / DeltaMilliSec);
    digitalWrite(ledPin, LOW);
    delay(500 - 400 * DeltaT / DeltaMilliSec);
    digitalWrite(ledPin, HIGH);
  }
}

float wheelPosRight = 0; //initial wheel speed right
float wheelPosLeft = 0; //initial wheel speed left

// pins 3, 5, 6, 9, 10 and 11 are PWM pins.
// pins A0-A5 are analog reading pins
// driving
int dir1PinRight = 6; //A+
int dir2PinRight = 9; //A-
int dir1PinLeft = 5; //B+
int dir2PinLeft = 3; //B-
int timeTurning = 900;
int timeDriving = 1000;
int drivingSpeed = 200;
int turningSpeed = 250;

// light sensor
int light_sensor_out_pin = 10;
int light_sensor_in_pin = A0;
int valueLight = 0;
int pwmLight = 0;

// command block reading
const int CommandBlockOut = 1;  // 1-13 are digital pins
const int CommandBlockIn1 = A1;
const int CommandBlockIn2 = A2;
const int CommandBlockIn3 = A3;

// for (running) average:
const int numReadingsLight = 10;

int readingsLight[numReadingsLight];     // the readings from the analog
input
int readIndexLight = 0;                  // the index of the current reading
```

```
int totalLight = 0;                         // the running total
int averageLight = 0;                       // the average
int totalLightTop = 0;                      // total of the light readings that
are above the total average
int averageLightTop = 0;                    // average of the light readings
that are above the total average

// everything between these two values means that the robot is on a grey
tile
int black_up = NULL;                           // upperbound for which we
decide that the robot is on a black tile
int white_low = NULL;                          // lowerbound for which we
decide that the robot is on a white tile

//Each type of command block will cause the arduino to read a different
voltage upon being used.
const double MoveForwardVolts = 507;
const double MoveBackwardVolts = 999;
const double RotateClockwiseVolts = 236;
const double RotateCounterClockwiseVolts = 317;
const double Loop2Volts = 90;
const double Loop3Volts = 679;
const double LoopStopVolts = 837;
const double CheckRange = 30;

//Command block commands will be represented with a constant integer type
const int MoveForwardCommand = 0;
const int MoveBackwardCommand = 1;
const int RotateClockwiseCommand = 2;
const int RotateCounterClockwiseCommand = 3;
const int Loop2Command = 4;
const int Loop3Command = 5;
const int LoopStopCommand = 6;
const int LoopAmount = 4;          // how many times should "loop" loop

//Everything related to the commands list and management thereof
const int CommandTotal = 20;
const int CommandChainTotal = 5 * CommandTotal;

int commands[CommandTotal];
int parsedCommandChain[CommandChainTotal];
int parsedIndex = 0;
int commandIndex = 0;
int commandsRead = 0;
int commandsParsed = 0;

//External human input and output, such as feedback lights and buttons
const int FeedbackLight = 13;
const int CommandButton = 12;
bool CommandButtonStatus = false;
const int ClearButton = 8;
bool ClearButtonStatus = false;
const int ExecuteButton = 2;
bool ExecuteButtonStatus = false;
const int buzzerButton = 11;       // PWM pin

void moveForward() {
  Serial.println("moving forward inside function");
  wheelPosRight = drivingSpeed;
  analogWrite(dir1PinRight, wheelPosRight);
  analogWrite(dir2PinRight, 0);
```

```
  wheelPosLeft = drivingSpeed;
  analogWrite(dir1PinLeft, wheelPosLeft);
  analogWrite(dir2PinLeft, 0);
  delay(timeDriving);
  moveStill();
}

void moveBackward() {
  wheelPosRight = drivingSpeed;
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, wheelPosRight);
  wheelPosLeft = drivingSpeed;
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, wheelPosLeft);
  delay(timeDriving);
  moveStill();
}

void moveLeft() {
  wheelPosRight = turningSpeed;
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, wheelPosRight);
  wheelPosLeft = turningSpeed;
  analogWrite(dir1PinLeft, wheelPosLeft);
  analogWrite(dir2PinLeft, 0);
  delay(timeTurning);
  moveStill();
}

void moveRight() {
  wheelPosRight = turningSpeed;
  analogWrite(dir1PinRight, wheelPosRight);
  analogWrite(dir2PinRight, 0);
  wheelPosLeft = turningSpeed;
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, wheelPosLeft);
  delay(timeTurning);
  moveStill();
}

// Stop moving
void moveStill() {
  wheelPosLeft = 0;
  wheelPosRight = 0;
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, 0);
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, 0);
}

//Given a boolean to discern whether to calculate for a black tile or white
tile, calculate black_high and white_low
void calculateColourRange(bool black) {
    for (int i = 0; i < numReadingsLight; i++) {
    valueLight = analogRead(light_sensor_in_pin);   // read the light
sensor input pin
    //Serial.print("light = ");
    //Serial.println(valueLight);                        // show value, this
will be between 0 and 1023 which represents 0-5 volt
```

```
    totalLight = totalLight - readingsLight[readIndexLight];      //
subtract the last reading:
    readingsLight[readIndexLight] = valueLight;                   // read
from the sensor:
    totalLight = totalLight + readingsLight[readIndexLight];      // add
the reading to the total:
    readIndexLight = readIndexLight + 1;                          //
advance to the next position in the array:

    if (readIndexLight >= numReadingsLight) {                     // end of
array is found
      readIndexLight = 0;                                         // wrap
around to the beginning:
    }
    delay(1);                                                    // some
delay to add stability to measurements
  }

  // The running best number that we have encountered so far.
  int searchLight;
  if (black) {
    searchLight = -1; //All read values should be positive and thus > -1
  } else {
    searchLight = 9999; // Arbitrarily high number, since Infinity doesn't
exist, is fine so long as any read value is definitely lower.
  }

  // loop through all readings and find the highest/lowest value
respectively.
  for (int i = 0; i < numReadingsLight; i++) {
    if (black) {
      if (readingsLight[i] > searchLight) {
        searchLight = readingsLight[i];
      }
    } else {
      if (readingsLight[i] < searchLight) {
        searchLight = readingsLight[i];
      }
    }
  }

  //Set the found value to the corresponding variable
  if (black) {
    Serial.print("Setting black_up to: ");
    Serial.println(searchLight);
    black_up = searchLight;
  } else {
    Serial.print("Setting white_low to: ");
    Serial.println(searchLight);
    white_low = searchLight;
  }
}

// return 0 if black tile, 1 if grey tile, 2 if white tile
int readLight() {
  for (int i = 0; i < numReadingsLight; i++) {
    valueLight = analogRead(light_sensor_in_pin);   // read the light
sensor input pin
    Serial.print("light = ");
    Serial.println(valueLight);                        // show value, this
will be between 0 and 1023 which represents 0-5 volt
```

```
    totalLight = totalLight - readingsLight[readIndexLight];       //
subtract the last reading:
    readingsLight[readIndexLight] = valueLight;                    // read
from the sensor:
    totalLight = totalLight + readingsLight[readIndexLight];       // add
the reading to the total:
    readIndexLight = readIndexLight + 1;                           //
advance to the next position in the array:

    if (readIndexLight >= numReadingsLight) {                      // end of
array is found
      readIndexLight = 0;                                          // wrap
around to the beginning:
    }
    delay(1);                                                    // some
delay to add stability to measurements
  }
  averageLight = totalLight / numReadingsLight;                  //
calculate the average

  // now we only want to use the read values that are above average
  int lightAmount = 0;                                           // keep
track how many elements are above average
  totalLightTop = 0;                                             // keep
track of total light (0 before start)
  for (int i = 0; i < numReadingsLight; i++) {                   // loop
through all readings and find the average of all elements that are above
average
    if (readingsLight[i] >= averageLight) {
      lightAmount++;
      totalLightTop = totalLightTop + valueLight;
    }
  }

  averageLightTop = totalLightTop / lightAmount;

  //Print out the values for debugging purposes.
  Serial.print("average = ");
  Serial.println(averageLight);
  Serial.print("average top= ");
  Serial.println(averageLightTop);

  //Decide which tile we're on
  if (averageLightTop < black_up) {                      // tile is black
    Serial.println("Tile is black");
    return 0;
  } else if (averageLightTop > white_low) {              // tile is white
    Serial.println("Tile is white");
    return 2;
  } else {
    return 1;                                         // tile is grey
    Serial.println("Tile is grey");
  }
}

// Buzz for long continuous moment
void blackTile() {
  Serial.println("On a black tile right now so I will buzz nonstop");
  analogWrite(buzzerButton, 5);                         // buzzer makes
sound
```

```
  delay(10000);
  analogWrite(buzzerButton, 0);
}

// Buzz intermittently for a few seconds
void greyTile() {
  Serial.println("On a grey tile right now so I will buzzer frequently");
  for (int i = 0; i < 10; i++) {
    analogWrite(buzzerButton, 5);
    delay(500);
    analogWrite(buzzerButton, 0);
    delay(500);
  }
}

//Don't take any extra actions.
void whiteTile() {
  Serial.println("On a white tile right now so I will do nothing");
}

//Read all three command block inputs
void readCommands() {
  readCommand(CommandBlockIn1, CommandBlockOut);
  readCommand(CommandBlockIn2, CommandBlockOut);
  readCommand(CommandBlockIn3, CommandBlockOut);
}

// Sets the output for the CommandOut pin to high.
// Then reads the resulting value to CommandIn to recognize the inputted
block.
// Finally sets output for CommandOut back to low.
void readCommand(int CommandIn, int CommandOut) {
  digitalWrite(FeedbackLight, HIGH);
  digitalWrite(CommandOut, HIGH);
  double commandRead = analogRead(CommandIn);
  Serial.print("commandRead = ");
  Serial.println(commandRead);
  if (commandRead > MoveForwardVolts - CheckRange && commandRead <
MoveForwardVolts + CheckRange) {
    storeCommand(MoveForwardCommand);
    Serial.println("command read: MoveForwardCommand");
  } else if (commandRead > MoveBackwardVolts - CheckRange && commandRead <
MoveBackwardVolts + CheckRange) {
    storeCommand(MoveBackwardCommand);
    Serial.println("command read: MoveBackwardCommand");
  } else if (commandRead > RotateClockwiseVolts - CheckRange && commandRead
< RotateClockwiseVolts + CheckRange) {
    storeCommand(RotateClockwiseCommand);
    Serial.println("command read: RotateClockwiseCommand");
  } else if (commandRead > RotateCounterClockwiseVolts - CheckRange &&
commandRead < RotateCounterClockwiseVolts + CheckRange) {
    storeCommand(RotateCounterClockwiseCommand);
    Serial.println("command read: RotateCounterClockwiseCommand");
  } else if (commandRead > Loop2Volts - CheckRange && commandRead <
Loop2Volts + CheckRange) {
    storeCommand(Loop2Command);
    Serial.println("command read: Loop2Command");
  } else if (commandRead > Loop3Volts - CheckRange && commandRead <
Loop3Volts + CheckRange) {
    storeCommand(Loop3Command);
    Serial.println("command read: Loop3Command");
```

```
  } else if (commandRead > LoopStopVolts - CheckRange && commandRead <
LoopStopVolts + CheckRange) {
    storeCommand(LoopStopCommand);
    Serial.println("command read: LoopStopCommand");
  }
  digitalWrite(CommandOut, LOW);
  digitalWrite(FeedbackLight, LOW);
}

// Stores the resulting command in the commands array. Also updates
commandIndex and commandsRead accordingly.
void storeCommand(int command) {
  if (commandIndex == CommandTotal) {
    Serial.println("Command can not be stored. Maximum amount reached.");
    return;
  } else {
    commands[commandIndex] = command;
    commandIndex++;
    commandsRead++;
  }
}

// Empties all commands currently stored. Also updates commandIndex and
commandsRead to 0
void clearCommands() {
  for (int i = 0; i < CommandTotal; i++) {
    commands[i] = -1;
  }
  commandIndex = 0;
  commandsRead = 0;

  for (int i = 0; i < CommandChainTotal; i++) {
    parsedCommandChain[i] = -1;
  }
  parsedIndex = 0;
  commandsParsed = 0;
}

//Check the different effects of various PWM values
void PWMTest(int value) {
  analogWrite(PWMDebugPin, value);
  delay(PWMDebugTime);
  analogWrite(PWMDebugPin, 0);
}

//Turn the led on or off manually.
void toggleLight() {
  if (LightDebugStatus) {
    digitalWrite(ledPin, LOW);
  } else {
    digitalWrite(ledPin, HIGH);
  }
  LightDebugStatus = !LightDebugStatus;
  Serial.print("LightDebugStatus = ");
  Serial.println(LightDebugStatus);
}

//Manually re-calibrate the light sensor
void setCalibrationValues() {
  white_low = NULL;
  black_up = NULL;
```

```
  while (white_low == NULL || black_up == NULL) {
    if (Serial.available() > 0) {
      int inByte = Serial.read();
      int value = 0;
      switch (inByte) {
        case'b':
          value = Serial.parseInt();
          black_up = value;
          break;
        case'w':
          value = Serial.parseInt();
          white_low = value;
          break;
      }
    }
  }
}

//A set of commands to test each function manually and seperately
void debugging() {
  int quit = 0;                         // keep track if we want to quit the
program
  while (quit == 0) {
    if (Serial.available() > 0) {
      int inByte = Serial.read();         // input character byte

      switch (inByte)
      {
        case'a':
          WaitAndBlink(2000);
          break;
        case'b':
          moveForward();
          break;
        case'c':
          moveBackward();
          break;
        case'd':
          moveRight();
          break;
        case'e':
          moveLeft();
          break;
        case'f':
          //int valueRandom = readLight();
          readLight();
          break;
        case'g':
          blackTile();
          break;
        case'h':
          greyTile();
          break;
        case'i':
          whiteTile();
          break;
        case'j':
          toggleLight();
          break;
        case'k':
        {
```

```
          int PWMvalue = Serial.parseInt();
          PWMTest(PWMvalue);
          break;
        }
        case'n':
          readCommands();
          break;
        case'o':
          calibrateLightSensor();
          break;
        case'p':
          parseRoute();
          executeRoute();
        case'q':
          debuggingMode = 0;
          quit = 1;
          break;
        case'r':
          setCalibrationValues();
      }

    }
  }
}

// setup() is only run once
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);                    // set up Serial with a bits per
second (baud) rate of 9600

  // driving
  pinMode(13, OUTPUT);
  pinMode(dir1PinRight, OUTPUT);
  pinMode(dir2PinRight, OUTPUT);
  pinMode(dir1PinLeft, OUTPUT);
  pinMode(dir2PinLeft, OUTPUT);
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, 0);
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, 0);

  // light sensor
  pinMode(light_sensor_out_pin, OUTPUT);
  analogWrite(light_sensor_out_pin, pwmLight);
  for (int thisReadingLight = 0; thisReadingLight < numReadingsLight;
thisReadingLight++) {   // fill all array entries with 0
    readingsLight[thisReadingLight] = 0;
  }

  // input blocks reading
  pinMode(CommandBlockOut, OUTPUT);
  pinMode(FeedbackLight, OUTPUT);
  pinMode(CommandButton, INPUT);
  pinMode(ClearButton, INPUT);
  pinMode(ExecuteButton, INPUT);
  pinMode(buzzerButton, OUTPUT);

  //Initialize all outputs to 0
  digitalWrite(ledPin, LOW);
  digitalWrite(FeedbackLight, LOW);
```

```
  digitalWrite(CommandBlockOut, LOW);
  analogWrite(buzzerButton, 0);

  //Debug initialization
  pinMode(PWMDebugPin, OUTPUT);
  analogWrite(PWMDebugPin, 0);

  //Empty any commands in the list and finish
  clearCommands();
  //If you're not debugging, calibrate the light sensor
  if (debuggingMode != 1) {
    calibrateLightSensor();
  }
  Serial.println("Setup done");

//  WaitAndBlink(2000);                                                  //
show that setup is done


}

void calibrateLightSensor() {
  digitalWrite(FeedbackLight, HIGH);
  // Make players place the robot on a black tile and press the execute
button to read black_high
  int buttonRead;
  while (black_up == NULL) {
    buttonRead = digitalRead(ExecuteButton);
    if (buttonRead == HIGH && !ExecuteButtonStatus) {
      calculateColourRange(true);
      ExecuteButtonStatus = true;
    } else if (buttonRead == LOW) {
      ExecuteButtonStatus = false;
    }
  }

  // Let players know that black_up has been calculated
  analogWrite(buzzerButton, 5);         // Preferably, this becomes a
light, rather than a buzzer.
  delay(1000);
  analogWrite(buzzerButton, 0);
  //Now do the same but for a white tile and white_low instead.
  while (white_low == NULL) {
    buttonRead = digitalRead(ExecuteButton);
    if (buttonRead == HIGH && !ExecuteButtonStatus) {
      calculateColourRange(false);
      ExecuteButtonStatus = true;
    } else if (buttonRead == LOW) {
      ExecuteButtonStatus = false;
    }
  }
  delay(50);
  analogWrite(buzzerButton, 5);
  delay(1000);
  analogWrite(buzzerButton, 0);
  digitalWrite(FeedbackLight, LOW);
}

// the commands in loop() are repeated forever
void loop() {
  if (debuggingMode == 1) {
```

```
    Serial.println("Going into debugging mode");
    debugging();
  }

  // Read the clear button
  int buttonRead = digitalRead(ClearButton);
  if (buttonRead == HIGH && !ClearButtonStatus) {
    clearCommands();
    ClearButtonStatus = true;
  } else if (buttonRead == LOW) {
    ClearButtonStatus = false;
  }

  //Read the command-reading button
  buttonRead = digitalRead(CommandButton);
  if (buttonRead == HIGH && !CommandButtonStatus) {
    readCommands();
    CommandButtonStatus = true;
  } else if (buttonRead == LOW){
    CommandButtonStatus = false;
  }

  //Read the execute button
  buttonRead = digitalRead(ExecuteButton);
  if (buttonRead == HIGH && !ExecuteButtonStatus) {
    parseRoute();
    delay(1500);
    executeRoute();
    ExecuteButtonStatus = true;
  } else if (buttonRead == LOW) {
    ExecuteButtonStatus = false;
  }
}

//Parse the entire route into a single executable chain.
void parseRoute() {
  Serial.println("Start parsing route");
  parsedIndex = 0;
  for (int i = 0; i < commandsRead; i++) {
    int currentCommand = commands[i];
    Serial.print("Parsing ");
    Serial.println(currentCommand);

    //Commands: Forward = 0, Backward = 1, Clockwise/Right = 2,
CounterClockwise/Left = 3, BeginLoop2 = 4, Beginloop3 = 5, Stoploop = 6
    switch (currentCommand) {
      case MoveForwardCommand:
      case MoveBackwardCommand:
      case RotateClockwiseCommand:
      case RotateCounterClockwiseCommand:
        if (commandsParsed < CommandChainTotal) {
          parsedCommandChain[parsedIndex] = currentCommand;
          parsedIndex++;
        } else {
          Serial.print("Too many commands parsed");
        }
        commandsParsed++;
        break;
      case Loop2Command:
        i = parseLoop(i+1, 2) - 1; // -1 to offset the incrementing of i at
the end of the loop due to the nature of for-loops
```

```
        break;
      case Loop3Command:
        i = parseLoop(i+1, 3) - 1;
        break;
      case LoopStopCommand:
        break;
      default:
        Serial.println("This really shouldn't happen, handle this
somehow?");
    }
  }
  Serial.print("Current command chain: ");
  printParsedCommands();
}

//Print the entire parsed command chain for debugging purposes
void printParsedCommands() {
  for (int i = 0; i < commandsParsed; i++) {
    Serial.print(parsedCommandChain[i]);
    Serial.print(", ");
  }
  Serial.println();
}

// Parse all commands inside a loop
int parseLoop(int firstCommand, int loopAmount) {
  Serial.print("Parsing a loop ");
  Serial.print(loopAmount);
  Serial.print(". With firstCommand index ");
  Serial.println(firstCommand);
  int loopCount = 0;
  int i = firstCommand;
  while (loopCount < loopAmount) {
    Serial.println("Starting while loop");
    int currentCommand;
    if (i < CommandTotal) {
      currentCommand = commands[i];
    } else {
      currentCommand = LoopStopCommand;
    }

    Serial.print("Read ");
    Serial.print(currentCommand);
    Serial.println(" in loop");

    switch (currentCommand) {
      case MoveForwardCommand:
      case MoveBackwardCommand:
      case RotateClockwiseCommand:
      case RotateCounterClockwiseCommand:
        if (commandsParsed < CommandChainTotal) {
          parsedCommandChain[parsedIndex] = currentCommand;
          parsedIndex++;
        } else {
          Serial.print("Too many commands parsed");
        }
        commandsParsed++;
        i++;
        break;
      case Loop2Command:
        i = parseLoop(i+1, 2);
```

```
        break;
      case Loop3Command:
        i = parseLoop(i+1, 3);
        break;
      default:
      case LoopStopCommand:
        loopCount++;
        if (loopAmount - loopCount > 0) {
          i = firstCommand;
        } else {
          i++;
        }
        break;
    }
  }
  return i;
}

//Execute the entire parsed command chain
void executeRoute() {
  Serial.println("Executing commands now");
  for (int i = 0; i < commandsParsed; i++)      // one for one go through
all commands
  {
    int currentCommand = parsedCommandChain[i]; // retrieve command
    Serial.print("Executing command: ");
    Serial.println(currentCommand);

    // commands: forward = 0, backward = 1, clockwise/right = 2,
counterclockwise/left = 3, beginloop2 = 4, stoploop = 5, beginloop3 = 6.
stand still = 9
    switch (currentCommand)
    {
      case MoveForwardCommand:
        Serial.println("Moving forward");
        moveForward();
        break;
      case MoveBackwardCommand:
        Serial.println("Moving backward");
        moveBackward();
        break;
      case RotateClockwiseCommand:
        Serial.println("Moving right");
        moveRight();
        break;
      case RotateCounterClockwiseCommand:
        Serial.println("Moving left");
        moveLeft();
        break;
      case '9':
        Serial.println("Standing still");
        moveStill();
        break;
      default:
        Serial.println("This isn't right, this shouldn't be!");
    }
    // read light to check if we are on a permitted square
    int lightReading = readLight();        // return 0 if black tile, 1 if
grey tile, 2 if white tile
    if (lightReading == 0) {
      blackTile();
```

```
      break;
    } else if (lightReading == 1) {
      greyTile();
      break;
    } else {
      whiteTile();
    }
  }
  Serial.println("Done driving");
  clearCommands();
}
```