```
#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 = 1;        // go into debugging/testing mode to check if
seperate modules work

int commandAvailable = 0;     // how many commands are available
// 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;
int dir2PinRight = 9;
int dir1PinLeft = 5;
int dir2PinLeft = 3;
int timeTurning = 100;
int timeDriving = 10000;

// 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 CommandBlockIn = A1;

// 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 = 0;                       // upperbound for which we decide
that the robot is on a black tile
int white_low = 0;                      // 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 = 1.0;
const double MoveBackwardVolts = 2.0;
const double RotateClockwiseVolts = 3.0;
const double RotateCounterClockwiseVolts = 4.0;
const double LoopVolts = 5.0;
const double LoopStopVolts = 6.0;
const double CheckRange = 20;

//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 LoopCommand = 4;
const int LoopAmount = 4;         // how many times should "loop" loop
const int LoopStopCommand = 5;

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

int commands[CommandTotal];
int commandIndex = 0;

//External human input and output, such as feedback lights and buttons
const int FeedbackLight = 13;
const int CommandButton = 12;
const int ClearButton = 11;

void moveForward() {
  Serial.print("moving forward inside function");
  wheelPosRight = 250;
  analogWrite(dir1PinRight, wheelPosRight);
  analogWrite(dir2PinRight, 0);
  wheelPosLeft = 250;
  analogWrite(dir1PinLeft, wheelPosLeft);
  analogWrite(dir2PinLeft, 0);
  delay(timeDriving);
  moveStill();
  /*
    if (wheelPosLeft < 201) {
    wheelPosLeft = wheelPosLeft + 50;
    }
    if (wheelPosLeft >= 0 ) {
    analogWrite(dir1PinLeft, 0);
    analogWrite(dir2PinLeft, wheelPosLeft);
    } else {
    analogWrite(dir1PinLeft, abs(wheelPosLeft));
    analogWrite(dir2PinLeft, 0);
    }

    if (wheelPosRight < 201) {
```

```
    wheelPosRight = wheelPosRight + 50;
    }
    if (wheelPosRight >= 0 ) {
    analogWrite(dir1PinRight, 0);
    analogWrite(dir2PinRight, wheelPosRight);
    } else {
    analogWrite(dir1PinRight, abs(wheelPosRight));
    analogWrite(dir2PinRight, 0);
    }
  */
}

void moveBackward() {
  wheelPosRight = 250;
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, wheelPosRight);
  wheelPosLeft = 250;
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, wheelPosLeft);
  delay(timeDriving);
  moveStill();
  /*
    if (wheelPosLeft >= -201) {
    wheelPosLeft = wheelPosLeft - 50;
    }
    if (wheelPosLeft >= 0 ) {
    analogWrite(dir1PinLeft, 0);
    analogWrite(dir2PinLeft, wheelPosLeft);
    } else {
    analogWrite(dir1PinLeft, abs(wheelPosLeft));
    analogWrite(dir2PinLeft, 0);
    }

    if (wheelPosRight >= -201) {
    wheelPosRight = wheelPosRight - 50;
    }
    if (wheelPosRight >= 0 ) {
    analogWrite(dir1PinRight, 0);
    analogWrite(dir2PinRight, wheelPosRight);
    } else {
    analogWrite(dir1PinRight, abs(wheelPosRight));
    analogWrite(dir2PinRight, 0);
    }
  */
}

void moveLeft() {
  wheelPosRight = 250;
  analogWrite(dir1PinRight, 0);
  analogWrite(dir2PinRight, wheelPosRight);
  wheelPosLeft = 250;
  analogWrite(dir1PinLeft, wheelPosLeft);
  analogWrite(dir2PinLeft, 0);
  delay(timeTurning);
  moveStill();
}

void moveRight() {
  wheelPosRight = 250;
  analogWrite(dir1PinRight, wheelPosRight);
  analogWrite(dir2PinRight, 0);
  wheelPosLeft = 250;
  analogWrite(dir1PinLeft, 0);
  analogWrite(dir2PinLeft, wheelPosLeft);
  delay(timeTurning);
  moveStill();
}

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

// 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;
  Serial.print("average = ");
  Serial.println(averageLight);
  Serial.print("average top= ");
  Serial.println(averageLightTop);

  if (averageLightTop < black_up) {                    // tile is black
    return 0;
  } else if (averageLightTop > white_low) {            // tile is white
    return 2;
  } else {
    return 1;                                       // tile is grey
  }
}

void blackTile() {
  // do some stuff if we are on a black tile
}

void greyTile() {
  // do some stuff if we are on a grey tile
}

void whiteTile() {
  // do some stuff if we are on a white tile
}

void readCommands() {
  readCommand(CommandBlockIn, CommandBlockOut);
}

// Sets the output for the CommandOut pin to high.
// Then reads the resulting value to CommandIn to recognize the inputted
block.
void readCommand(int CommandIn, int CommandOut) {
  digitalWrite(FeedbackLight, HIGH);
  digitalWrite(CommandOut, HIGH);
  double commandRead = analogRead(CommandIn);

  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 > LoopVolts - CheckRange && commandRead <
LoopVolts + CheckRange) {
    storeCommand(LoopCommand);
    Serial.println("command read: LoopCommand");
  } 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 accordingly.
void storeCommand(int command) {
  if (commandIndex == CommandTotal) {
    Serial.print("Command can not be stored. Maximum amount reached.");
    return;
  } else {
    commands[commandIndex] = command;
    commandIndex++;
  }
}

// Undoes the last command stored. Also updates the commandIndex. Returns
the command for robustness
int undoCommandRead() {
  if (commands[commandIndex] == NULL) {
    Serial.print("Command can not be undone. No command stored");
  } else {
    commandIndex--;
    int undoneCommand = commands[commandIndex];
    commands[commandIndex] = NULL;
    return undoneCommand;
  }
}

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

// Returns the entire commands list. Does nothing else currently.
// Apparently not possible in Arduino, keeping it here just in case.
//int[] retrieveAllCommands() {
//  return commands;
//}

// Retrieves the first stored command in the list
// Returns the command if one is found, null if no command is found.
int retrieveFirstCommand() {
  commandIndex = 0;

  while (commands[commandIndex] == NULL && commandIndex < CommandTotal) {
    commandIndex++;
  }
  if (commandIndex == CommandTotal) {
    Serial.print("Command can not be retrieved. No command stored");
    return NULL;
  } else {
    int retrievedCommand = commands[commandIndex];
    commands[commandIndex] = NULL;
    Serial.print("Retrieved command: ");
    Serial.println(retrievedCommand);
    return retrievedCommand;
  }
}

void doSomething() {

  analogWrite(13, HIGH);
  analogWrite(dir2PinRight, LOW);
}

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':
        Serial.println("going to move forward");
        doSomething();
         // moveForward();

          break;
        case'c':
          moveBackward();
          break;
        case'd':
          moveRight();
          break;
        case'e':
          moveLeft();
          break;
        case'f':
          int valueRandom = readLight();
          break;
        case'g':
          blackTile();
          break;
        case'h':
          greyTile();
          break;
        case'i':
          whiteTile();
          break;
        case'q':
          quit = 1;
          break;
      }

    }
  }
}

// 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);

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

  //Empty any commands in the list and finish
  clearCommands();

  Serial.println("Setup done");

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

// the commands in loop() are repeated forever
void loop() {
  if (debuggingMode == 1) {
    Serial.println("Going into debugging mode");
    debugging();
  }


  int buttonRead = digitalRead(ClearButton);
  if (buttonRead == HIGH) {
    clearCommands();
  } else {
    buttonRead = digitalRead(CommandButton);
    if (buttonRead == HIGH) {
      readCommands();
    }
  }
  for (int i = 0; i < CommandTotal; i++)      // one for one go through all
commands
  {
    int currentCommand = commands[i]; // retrieve command

    // commands: forward = 0, backward = 1, clockwise/right = 2,
counterclockwise/left = 3, beginloop = 4, stoploop = 5. stand still = 9
    switch (currentCommand)
    {
      case'0':
        moveForward();
        break;
      case'1':
        moveBackward();
        break;
      case'2':
        moveRight();
        break;
      case'3':
        moveLeft();
        break;
      case'4':
        int commandsInsideLoop = 0;           // keep track of how many
commands are inside the loop so we know how many steps back we have to
begin again
        for (int y = 0; y < LoopAmount; y++) {
          commandsInsideLoop = 0;
          int currentCommand = commands[i + 1]; // retrieve command
          while (currentCommand != 5) {
            currentCommand = commands[i + 1];     // retrieve command
            // commands: forward = 0, backward = 1, clockwise/right = 2,
counterclockwise/left = 3, beginloop = 4, stoploop = 5. stand still = 9
            if ( currentCommand == 0) {
              moveForward();
            } else if ( currentCommand == 1) {
              moveBackward();
            } else if ( currentCommand == 2) {
              moveRight();
            } else if ( currentCommand == 3) {
              moveLeft();
            }

            commandsInsideLoop++;
            i++;
          }
          i = i - commandsInsideLoop;
        }
        i = i + commandsInsideLoop;
        i++;
        break;
      case '9':
        moveStill();
        break;
    }
    // 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();
    } else if (lightReading == 1) {
      greyTile();
    } else {
      whiteTile();
    }
  }
}
```