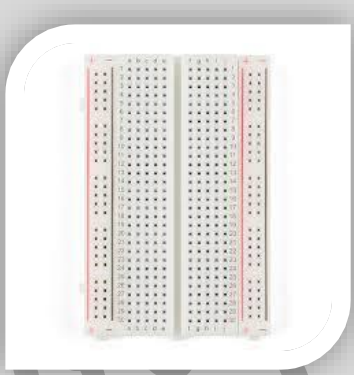
Arduino snake game

The idea of the project

The idea is to make a snake game using LEDs.

MATERIALS

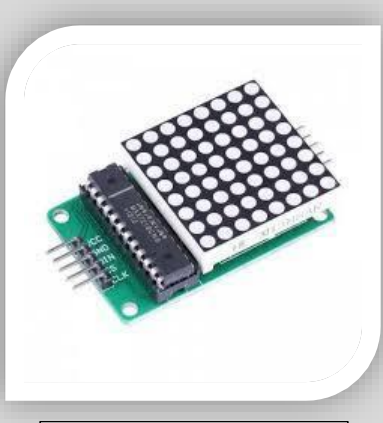
Arduino



Bread board

wires

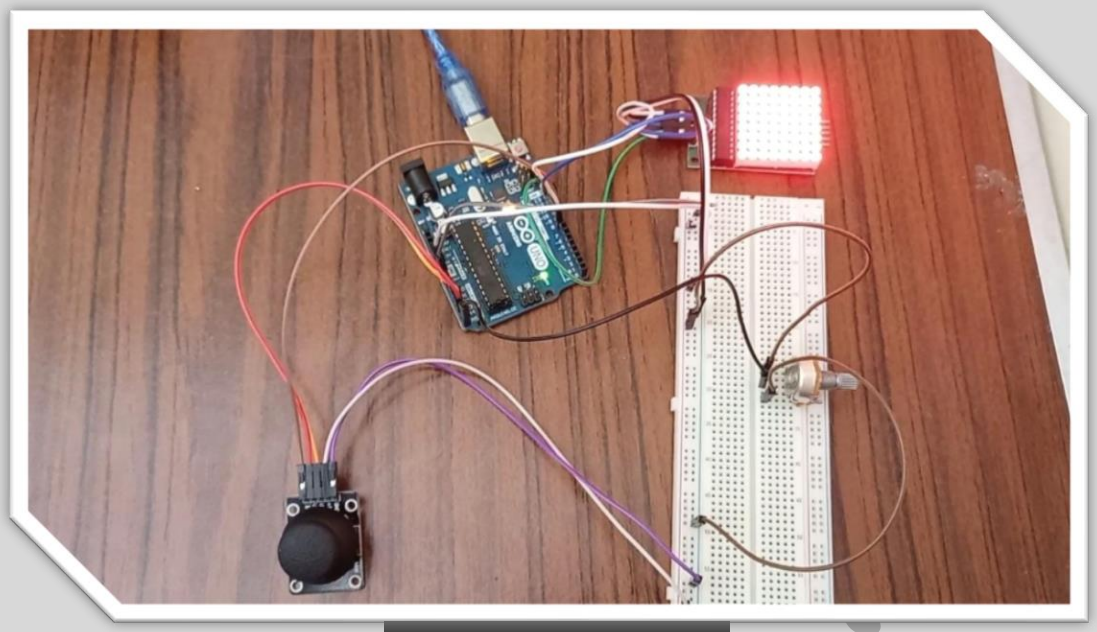
joystick



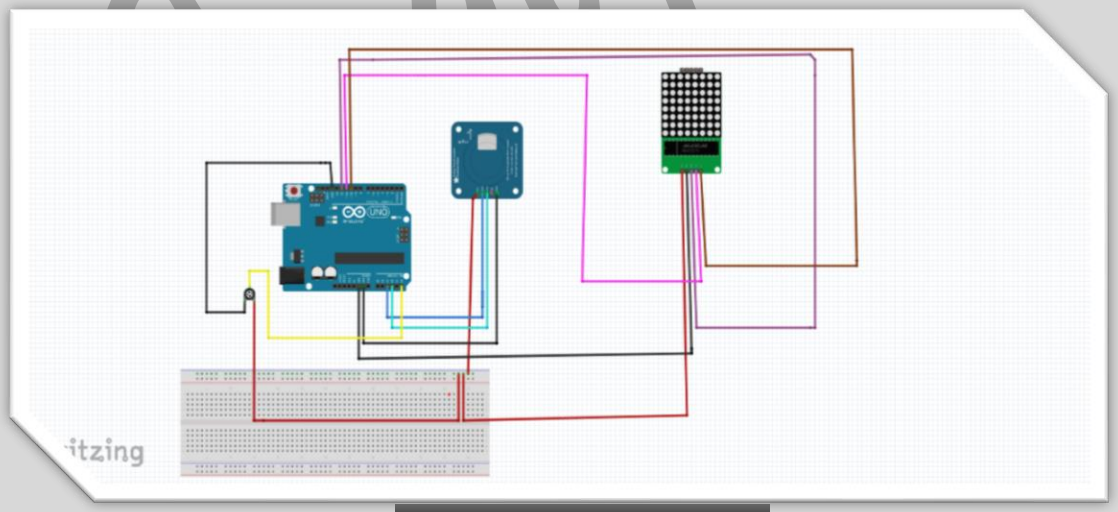
matrix

potentiometer

OUR PROTOTYPE



Our prototype in Real life



simulation to Our prototype

CODE

```
#include "LedControl.h"
struct Pin {
  static const short joystickX = A2;
  static const short joystickY = A3;
  static const short joystickVCC = 1;
  static const short joystickGND = 2;

  static const short potentiometer = A5;

  static const short CLK = 8;
  static const short CS  = 9;
  static const short DIN = 10;
};

// LED matrix brightness: between 0(darkest) and 15(brightest)
const short intensity = 8;

const short messageSpeed = 5;

const short initialSnakeLength = 3;


void setup() {
  Serial.begin(115200);
  initialize();
```

Group B

```
  calibrateJoystick();
  showSnakeMessage();
}



void loop() {
  generateFood();
  scanJoystick();
  calculateSnake();
  handleGameStates();

  // uncomment this if you want the current game board to be printed to the serial (slows down the game a bit)
  // dumpGameBoard();
}




// ------------------------------------------------------------ //
// ------------------- supporting variables ------------------- //
// ------------------------------------------------------------ //

LedControl matrix(Pin::DIN, Pin::CLK, Pin::CS, 1);

struct Point {
  int row = 0, col = 0;
  Point(int row = 0, int col = 0): row(row), col(col) {}
```

Group B

```
};

struct Coordinate {
  int x = 0, y = 0;
  Coordinate(int x = 0, int y = 0): x(x), y(y) {}
};

bool win = false;
bool gameOver = false;

// primary snake head coordinates (snake head), it will be randomly generated
Point snake;

// food is not anywhere yet
Point food(-1, -1);

// construct with default values in case the user turns off the calibration
Coordinate joystickHome(500, 500);

// snake parameters
int snakeLength = initialSnakeLength;
int snakeSpeed = 1;
int snakeDirection = 0;

// direction constants
const short up    = 1;
const short right  = 2;
const short down   = 3;
const short left   = 4;
```

Group B

```
const int joystickThreshold = 160;


const float logarithmity = 0.4;

int gameboard[8][5] = {};




// ------------------------------------------------------------- //
// ------------------------- functions ------------------------- //
// ------------------------------------------------------------- //


// if there is no food, generate one, also check for victory
void generateFood() {
  if (food.row == -1 || food.col == -1) {
    // self-explanatory
    if (snakeLength >= 64) {
      win = true;
      return;
    }


    do {
      food.col = random(5);
```

Group B

```
    food.row = random(8);
   } while (gameboard[food.row][food.col] > 0);
  }
}

void scanJoystick() {
  int previousDirection = snakeDirection;
  long timestamp = millis();

  while (millis() < timestamp + snakeSpeed) {
    float raw = mapf(analogRead(Pin::potentiometer), 0, 1023, 0, 1);
   snakeSpeed = mapf(pow(raw, 3.5), 0, 1, 10, 1000);
   if (snakeSpeed == 0) snakeSpeed = 1;

   analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold ? snakeDirection = up    : 0;
   analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold ? snakeDirection = down  : 0;
   analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold ? snakeDirection = left  : 0;
   analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold ? snakeDirection = right : 0;
   snakeDirection + 2 == previousDirection && previousDirection != 0 ? snakeDirection =
previousDirection : 0;
   snakeDirection - 2 == previousDirection && previousDirection != 0 ? snakeDirection =
previousDirection : 0;
   matrix.setLed(0, food.row, food.col, millis() % 100 < 50 ? 1 : 0);
  }
}
void calculateSnake() {
  switch (snakeDirection) {
   case up:
    snake.row--;
```

Group B

```
    fixEdge();
    matrix.setLed(0, snake.row, snake.col, 1);
    break;

  case right:
    snake.col++;
    fixEdge();
    matrix.setLed(0, snake.row, snake.col, 1);
    break;

  case down:
    snake.row++;
    fixEdge();
    matrix.setLed(0, snake.row, snake.col, 1);
    break;

  case left:
    snake.col--;
    fixEdge();
    matrix.setLed(0, snake.row, snake.col, 1);
    break;

  default:
    return;
}
if (gameboard[snake.row][snake.col] > 1 && snakeDirection != 0) {
  gameOver = true;
  return;
}
```

Group B

```
  if (snake.row == food.row && snake.col == food.col) {
    food.row = -1;
    food.col = -1;
    snakeLength++;
    for (int row = 0; row < 8; row++) {
      for (int col = 0; col < 5; col++) {
        if (gameboard[row][col] > 0 ) {
          gameboard[row][col]++;
        }
      }
    }
  }
  gameboard[snake.row][snake.col] = snakeLength + 1;
  for (int row = 0; row < 8; row++) {
    for (int col = 0; col < 5; col++) {
      if (gameboard[row][col] > 0 ) {
        gameboard[row][col]--;
      }
      matrix.setLed(0, row, col, gameboard[row][col] == 0 ? 0 : 1);
    }
  }
}
void fixEdge() {
  snake.col < 0 ? snake.col += 5 : 0;
  snake.col > 4 ? snake.col -= 5 : 0;
  snake.row < 0 ? snake.row += 8 : 0;
  snake.row > 7 ? snake.row -= 8 : 0;
}
void handleGameStates() {
```

Group B

```
  if (gameOver || win) {
    unrollSnake();
    showScoreMessage(snakeLength - initialSnakeLength);
    if (gameOver) showGameOverMessage();
    else if (win) showWinMessage();
    win = false;
    gameOver = false;
    snake.row = random(8);
    snake.col = random(5);
    food.row = -1;
    food.col = -1;
    snakeLength = initialSnakeLength;
    snakeDirection = 0;
    memset(gameboard, 0, sizeof(gameboard[0][0]) * 8 * 5);
    matrix.clearDisplay(0);
  }
}
void unrollSnake() {
  matrix.setLed(0, food.row, food.col, 0);
  delay(800);
  for (int i = 0; i < 5; i++) {
    for (int row = 0; row < 8; row++) {
      for (int col = 0; col < 5; col++) {
        matrix.setLed(0, row, col, gameboard[row][col] == 0 ? 1 : 0);
      }
    }
    delay(20);

    for (int row = 0; row < 8; row++) {
```

Group B

```
    for (int col = 0; col < 5; col++) {
      matrix.setLed(0, row, col, gameboard[row][col] == 0 ? 0 : 1);
    }
   }
   delay(50);
  }
  delay(600);
  for (int i = 1; i <= snakeLength; i++) {
   for (int row = 0; row < 8; row++) {
    for (int col = 0; col < 5; col++) {
     if (gameboard[row][col] == i) {
      matrix.setLed(0, row, col, 0);
      delay(100);
     }
    }
   }
  }
}

// contuince from here for norhan
// calibrate the joystick home for 10 times
void calibrateJoystick() {
  Coordinate values;

  for (int i = 0; i < 10; i++) {
   values.x += analogRead(Pin::joystickX);
   values.y += analogRead(Pin::joystickY);
  }
```

Group B

```
  joystickHome.x = values.x / 10;
  joystickHome.y = values.y / 10;
}


void initialize() {
  pinMode(Pin::joystickVCC, OUTPUT);
  digitalWrite(Pin::joystickVCC, HIGH);

  pinMode(Pin::joystickGND, OUTPUT);
  digitalWrite(Pin::joystickGND, LOW);

  matrix.shutdown(0, false);
  matrix.setIntensity(0, intensity);
  matrix.clearDisplay(0);

  randomSeed(analogRead(A2));
  snake.row = random(8);
  snake.col = random(5);
}


void dumpGameBoard() {
  String buff = "\n\n\n";
  for (int row = 0; row < 8; row++) {
    for (int col = 0; col < 5; col++) {
      if (gameboard[row][col] < 10) buff += " ";
      if (gameboard[row][col] != 0) buff += gameboard[row][col];
      else if (col == food.col && row == food.row) buff += "@";
```

Group B

```
      else buff += "-";
      buff += " ";
    }
    buff += "\n";
  }
  Serial.println(buff);
}
```

```
// ---------------------------------------------------------- //
// ------------------------- messages ------------------------- //
// ---------------------------------------------------------- //

const PROGMEM bool snakeMessage[8][84] = {
  {0,0,0,0,0,0,0,0, 0,1,1,0,0,0,0,1,1,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,1,0,0,1,1,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,0,1,1,0,1,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,0,1,1,0,1,1,0,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,0,1,1,0,1,1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0,
1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,0,1,1,0,1,1,0,1,1,0,1,0,1,0,0,0,0,0,0,0,0,0,0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0,0,0,0,0,0,0,0, 0,1,1,0,0,0,0,1,1,0,1,1,0,0,1,0,0,0,0,1,1,0,0,0,1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

Group B

```
  {0,0,0,0,0,0,0,0, 0,1,1,0,0,0,0,1,1,0,1,1,0,0,0,1,0,0,0,1,1,0,0,0,1, 0,1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0}

};


const PROGMEM bool gameOverMessage[8][90] = {

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0}

};


const PROGMEM bool scoreMessage[8][58] = {

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

Group B

```
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},

  {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

};


const PROGMEM bool digits[][8][8] = {

 {

   {0, 0, 0, 0, 0, 0, 0, 0},

   {0, 0, 1, 1, 1, 1, 0, 0},

   {0, 1, 1, 0, 0, 1, 1, 0},

   {0, 1, 1, 0, 1, 1, 1, 0},

   {0, 1, 1, 1, 0, 1, 1, 0},

   {0, 1, 1, 0, 0, 1, 1, 0},

   {0, 1, 1, 0, 0, 1, 1, 0},

   {0, 0, 1, 1, 1, 1, 0, 0}

 },

 {

   {0, 0, 0, 0, 0, 0, 0, 0},

   {0, 0, 0, 1, 1, 0, 0, 0},

   {0, 0, 0, 1, 1, 0, 0, 0},
```

Group B

```
  {0, 0, 1, 1, 1, 0, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0},
  {0, 1, 1, 1, 1, 1, 1, 0}
},
{
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 0, 0, 0, 1, 1, 0, 0},
  {0, 0, 1, 1, 0, 0, 0, 0},
  {0, 1, 1, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 1, 1, 1, 0}
},
{
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 0, 0, 1, 1, 1, 0, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0}
},
{
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 1, 1, 0, 0},
```

Group B

```
  {0, 0, 0, 1, 1, 1, 0, 0},
  {0, 0, 1, 0, 1, 1, 0, 0},
  {0, 1, 0, 0, 1, 1, 0, 0},
  {0, 1, 1, 1, 1, 1, 1, 0},
  {0, 0, 0, 0, 1, 1, 0, 0},
  {0, 0, 0, 0, 1, 1, 0, 0}
 },
 {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 1, 1, 1, 0},
  {0, 1, 1, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 1, 1, 0, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0}
 },
 {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 0, 0, 0},
  {0, 1, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0}
 },
 {
  {0, 0, 0, 0, 0, 0, 0, 0},
```

Group B

```
  {0, 1, 1, 1, 1, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 0, 0, 1, 1, 0, 0},
  {0, 0, 0, 0, 1, 1, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0},
  {0, 0, 0, 1, 1, 0, 0, 0}
 },
 {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0}
 },
 {
  {0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 1, 1, 1, 1, 0, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 1, 0},
  {0, 0, 0, 0, 0, 1, 1, 0},
  {0, 1, 1, 0, 0, 1, 1, 0},
  {0, 0, 1, 1, 1, 1, 0, 0}
 }
};
```

Group B

```
// scrolls the 'snake' message around the matrix
void showSnakeMessage() {
  [&] {
    for (int d = 0; d < sizeof(snakeMessage[0]) - 4; d++) {
      for (int col = 0; col < 5; col++) {
        delay(messageSpeed);
        for (int row = 0; row < 8; row++) {
          // this reads the byte from the PROGMEM and displays it on the screen
          matrix.setLed(0, row, col, pgm_read_byte(&(snakeMessage[row][col + d])));
        }
      }

      // if the joystick is moved, exit the message
      if (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
            || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
            || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
            || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {
        return; // return the lambda function
      }
    }
  }();

  matrix.clearDisplay(0);

  // wait for joystick co come back
  while (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
        || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
```

Group B

```
        || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
        || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {}

}


// scrolls the 'game over' message around the matrix
void showGameOverMessage() {
  [&] {
    for (int d = 0; d < sizeof(gameOverMessage[0]) - 4; d++) {
      for (int col = 0; col < 5; col++) {
        delay(messageSpeed);
        for (int row = 0; row < 8; row++) {
          // this reads the byte from the PROGMEM and displays it on the screen
          matrix.setLed(0, row, col, pgm_read_byte(&(gameOverMessage[row][col + d])));
        }
      }

      // if the joystick is moved, exit the message
      if (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
            || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
            || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
            || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {
        return; // return the lambda function
      }
    }
  }();

  matrix.clearDisplay(0);
```

Group B

```
  // wait for joystick co come back
  while (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
         || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
         || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
         || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {}

}


// scrolls the 'win' message around the matrix
void showWinMessage() {
  // not implemented yet // TODO: implement it
}


// scrolls the 'score' message with numbers around the matrix
void showScoreMessage(int score) {
  if (score < 0 || score > 99) return;

  // specify score digits
  int second = score % 10;
  int first = (score / 10) % 10;

  [&] {
    for (int d = 0; d < sizeof(scoreMessage[0]) + 2 * sizeof(digits[0][0]); d++) {
      for (int col = 0; col < 5; col++) {
        delay(messageSpeed);
        for (int row = 0; row < 8; row++) {
```

Group B

```
        if (d <= sizeof(scoreMessage[0]) - 8) {
          matrix.setLed(0, row, col, pgm_read_byte(&(scoreMessage[row][col + d])));
        }

        int c = col + d - sizeof(scoreMessage[0]) + 6; // move 6 px in front of the previous message

        // if the score is < 10, shift out the first digit (zero)
        if (score < 10) c += 8;

        if (c >= 0 && c < 8) {
          if (first > 0) matrix.setLed(0, row, col, pgm_read_byte(&(digits[first][row][c]))); // show only if score is >= 10 (see above)
        } else {
          c -= 8;
          if (c >= 0 && c < 8) {
            matrix.setLed(0, row, col, pgm_read_byte(&(digits[second][row][c]))); // show always
          }
        }
      }
    }

    // if the joystick is moved, exit the message
    if (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
          || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
          || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
          || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {
      return; // return the lambda function
    }
  }
```

Group B

```
  }();

  matrix.clearDisplay(0);

  //  // wait for joystick co come back
  //  while (analogRead(Pin::joystickY) < joystickHome.y - joystickThreshold
  //         || analogRead(Pin::joystickY) > joystickHome.y + joystickThreshold
  //         || analogRead(Pin::joystickX) < joystickHome.x - joystickThreshold
  //         || analogRead(Pin::joystickX) > joystickHome.x + joystickThreshold) {}

}


// standard map function, but with floats
float mapf(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Group B