//  The values info matrix represent the following;

//  mu1 is here == 1

//  mu2 is here == 2

//  mu3 is here == 3

//  BU is here == 4

//  This tile is scanned == 5

//  This tile has not been scanned == 0

//  The target is there == 6

//  mu is planning to go here == 7

//  obtacle here == 8

#include "DecisionUnArd\_header.h";

// Define positions

Position buPosition = {5, 5};

Position currentPosition = {6, 1};

// Position unknownTargetPosition = {1, 9};

Position targetPosition = {7,9};

// Define obstacles

Position obstacles[MAX\_OBSTACLES];

int obstacleCount = 2;

int buLink = 0;

// Define mu\_info\_matrix

int mu\_info\_matrix[GRID\_SIZE][GRID\_SIZE] = {

    {0, 0, 0, 0, 0, 0, 0, 0, 0},

    {0, 0, 0, 0, 8, 8, 0, 0, 0},

    {0, 0, 0, 0, 0, 0, 0, 0, 0},

    {7, 7, 7, 7, 7, 7, 7, 7, 7},

    {7, 7, 7, 7, 4, 7, 7, 7, 7},

    {7, 7, 7, 7, 7, 7, 7, 7, 7},

    {7, 7, 7, 7, 7, 7, 7, 7, 7},

    {7, 7, 7, 7, 7, 7, 7, 7, 7},

    {7, 7, 7, 7, 7, 7, 7, 7, 7}

};

// Set state to MOVING\_TO\_SEARCH\_AREA

int state = MOVING\_TO\_SEARCH\_AREA;

// Test function

// Test function

void test\_move\_decider\_MU1() {

    // Call move\_decider\_MU1

    Position nextMoveResult = move\_decider\_MU1(buPosition, currentPosition, targetPosition, obstacles, obstacleCount, mu\_info\_matrix, state, buLink);

    if(nextMoveResult.x != 0 || nextMoveResult.y != 0){

      // Checking first the BU connecition

      if(((buPosition.x-currentPosition.x)^2 + (buPosition.y-currentPosition.y)^2)<10){

        buLink = 1;

        Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*BU LINK IS ESTABLISHED");

      }

      else{

        buLink = 0;

        Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*NO LINK WITH THE BU");

      }

      // Updating the info matrix

      if(currentPosition.x == targetPosition.x && currentPosition.y == targetPosition.y){

      Serial.println("Target has been found.");

      state = TARGET\_FOUND;

        // Updating the info matrix

      mu\_info\_matrix[currentPosition.x-1][currentPosition.y - 1] = 6;

      }

      else{

        mu\_info\_matrix[currentPosition.x-1][currentPosition.y - 1] = 5;

      }

      currentPosition.x = currentPosition.x + nextMoveResult.x;

      currentPosition.y = currentPosition.y + nextMoveResult.y;

      mu\_info\_matrix[currentPosition.x-1][currentPosition.y - 1] = 1;

      // If our state is searching and we have not more unscaenned tiles left

      if(state == CURRENTLY\_SEARCHING && !unscannedTileLeft(mu\_info\_matrix)){

        state = TARGET\_NOT\_FOUND;

      }

    }

    // Output the result

    Serial.print("Move made: (");

    Serial.print(nextMoveResult.x);

    Serial.print(", ");

    Serial.print(nextMoveResult.y);

    Serial.println(")");

    // Current State

    Serial.print("Current State: ");

    Serial.println(state);

    // Current Position

    // Output the result

    Serial.print("Current Position: (");

    Serial.print(currentPosition.x);

    Serial.print(", ");

    Serial.print(currentPosition.y);

    Serial.println(")");

    // Print out the mu\_info\_matrix

    Serial.println("mu\_info\_matrix:");

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            Serial.print(mu\_info\_matrix[i][j]);

            Serial.print("\t");

        }

        Serial.println();

    }

}

// Setup function

void setup() {

    // Initialize serial communication

    Serial.begin(9600);

    obstacles[0].x = 2;

    obstacles[0].y = 5;

    obstacles[1].x = 2;

    obstacles[1].y = 6;

    // Test the move\_decider\_MU1 function

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    // Gets in the perimiter of the BU

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    Serial.println(buLink);

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    // Receives the info on the target from another MU

    state = MOVING\_TO\_TARGET;

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

    test\_move\_decider\_MU1();

}

// Loop function

void loop() {

    // Nothing to do here since we're testing in setup()

}

// Function to check if all elements in the matrix are non-zero

bool unscannedTileLeft(const int matrix[GRID\_SIZE][GRID\_SIZE]) {

    for (int i = 0; i < GRID\_SIZE; i++) {

        for (int j = 0; j < GRID\_SIZE; j++) {

            if (matrix[i][j] == 0) {

                return true;

            }

        }

    }

    return false;

}