C++ Project Folder\Robotic Project\robotMain.cpp

```cpp
#include <iostream>
#include "robotObject.hpp"
#include <ctime>
#include <vector>
#include <cstdlib>
#include <thread>
#include <chrono>
#include <algorithm>
// Macros Used in Program (pre-processors)
#define RESETVAR count = 0; moves = 0; fourDVUse = 0; rSensor = 0; dSensor = 0; uSensor = 0; lSensor = 0; secondStep1 = false; secondStep2 = false;
#define RESETSIM row = 0; column = 0; end = false;
#define FAILDETECT if(std::cin.fail() || std::cin.get() != '\n') {throw std::invalid_argument("Invalid input! Please enter an appropriate value.");}
#define SIMSTATS std::cout << "{SIMULATION STATISTICS} (can be inaccurate at times):\n\nMoves Counter: " << ((count == 0) ? moves - 1 : moves - 2) << "\nTotal Path Check (TPC): " << missed << "\n4DV Use Count: " << fourDVUse << "\nSensor Use Frequency:\t[Right Sensor -> " << rSensor << "]\n\t\t\t[Left Sensor -> " << lSensor << "]\n\t\t\t[Up Sensor -> " << uSensor << "]\n\t\t\t[Down Sensor -> " << dSensor << "]\n\n"; std::cout << "Robot Footprint (RED = initial, BLUE = footprint):\n"; showGeneralMatrix(traceMatrix);
#define CLEARANIMATION for(int d = 0; d < animatedMatrix.size(); d++) {for(int s = 0; s < animatedMatrix[0].size(); s++) {if(animatedMatrix[d][s].getName() == '+') {animatedMatrix[d][s] = Robot(' ', "");}}}
#define ANIMATE1 animatedMatrix[0][1] = Robot('+', "+"); animatedMatrix[1][0] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE2 animatedMatrix[0][2] = Robot('+', "+"); animatedMatrix[1][1] = Robot('+', "+"); animatedMatrix[2][0] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE3 animatedMatrix[0][3] = Robot('+', "+"); animatedMatrix[1][2] = Robot('+', "+"); animatedMatrix[2][1] = Robot('+', "+"); animatedMatrix[3][0] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE4 animatedMatrix[0][4] = Robot('+', "+"); animatedMatrix[1][3] = Robot('+', "+"); animatedMatrix[2][2] = Robot('+', "+"); animatedMatrix[3][1] = Robot('+', "+"); animatedMatrix[4][0] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE5 animatedMatrix[1][4] = Robot('+', "+"); animatedMatrix[2][3] = Robot('+', "+"); animatedMatrix[3][2] = Robot('+', "+"); animatedMatrix[4][1] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE6 animatedMatrix[2][4] = Robot('+', "+"); animatedMatrix[3][3] = Robot('+', "+"); animatedMatrix[4][2] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define ANIMATE7 animatedMatrix[3][4] = Robot('+', "+"); animatedMatrix[4][3] = Robot('+', "+"); showGeneralMatrix(animatedMatrix); repeat++;
#define DELAY loadingDelay(); loadingDelay();

// Variables
std::vector<std::vector<Robot>> matrix(5, std::vector<Robot>(5, Robot(' ', "")));
std::vector<std::vector<Robot>> animatedMatrix(5, std::vector<Robot>(5, Robot(' ', "")));
std::vector<std::vector<Robot>> traceMatrix(5, std::vector<Robot>(5, Robot(' ', "")));
char rName;
std::string rType;
int row = 0;
int column = 0;
```

```cpp
bool end = false;
int count = 0;
int moves = 0;
bool secondStep1 = false;
bool secondStep2 = false;
int repeat = 0;
int fourDVUse = 0;
int rSensor = 0;
int dSensor = 0;
int uSensor = 0;
int lSensor = 0;
int difficulty;
bool needMoreRound = true;
bool noPath = false;

// Function Prototypes
void createMatrix(std::vector<std::vector<Robot>>& matrix, std::vector<std::vector<Robot>>& traceMatrix, char rName, std::string rType, int difficulty);
void displayMatrix(std::vector<std::vector<Robot>>& matrix);
void updateMatrix(std::vector<std::vector<Robot>>& matrix);
bool scanObstacles(std::vector<std::vector<Robot>>& matrix);
void secondRound(std::vector<std::vector<Robot>>& matrix);
void loadingDelay();
void scanningAnimation();
void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix);
void showGeneralMatrix(std::vector<std::vector<Robot>>& generalMatrix);
int checkMissedPath(std::vector<std::vector<Robot>>& matrix);
int checkPerimeterPath(std::vector<std::vector<Robot>>& matrix);
void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix);
void clearTrace(std::vector<std::vector<Robot>>& traceMatrix);
void moveCursorUp(int lines);
void clearScreen();

// Runs all the main functions and text
int main()
{
    srand(time(NULL));
    std::cout << "\n\t\t\t\t\tRobotic Simulation\nPROGRAM USES ALGORITMIC SEQUENCES AND HEURISTICS TO MIMIC REAL LIFE ROBOTIC MOVEMENTS AND OBSTACLE DETECTION.\n\n\n";
    std::cout << "Simulation Success Rate (Tested 30 Times): 98%\n\n";
    std::cout << "Disclaimer: There might be pathways that you can see but the robot cannot. The robotic heuristic AND algorithm is not 100 percent accurate.\n";
    std::cout << "MAIN FEATURES: Active Robot Modern Scan (ARMS), Total Path Check (TPC), 4-way Directional Vision (4DV), Adaptive Double Attempt Logic (ADAL), Greedy Directional Navigation (GDN), Dynamic Remodification Algorithm (DRA), Obstacle Density Calibration (ODC), and Robot Footprint (RF).\n\n\n";
    std::cout << "This is a program that allows the robot to use ADAL and various other advanced algorithms to get through randomly placed obstacles to reach the end goal using obstacle avoidance.\n";
    std::cout << "However, the robot has 4 sensors for the 4 different direction. It mainly tries to go right or down (GDN), however, when it cannot, the robot will go left and up as
```

```
needed (4DV).\n\n\n";
    bool valid = false;
    do
    {
        try
        {
            std::cout << "Enter Robot Name (Single Character. Cannot be '0' or 'O'): ";
            std::cin >> rName;
            FAILDETECT
            while(rName == '0' || rName == 'O')
            {
                std::cout << "There is a naming conflict. The name cannot look similar to that of the obstacles for program validity. Try any other letter/number/symbol that is not '0' or 'O': ";
                std::cin >> rName;
            FAILDETECT
            }
            std::cout << "\nEnter difficulty (5 = less obstacles [super high success rate], 3 = mild obstacles [good success rate], 2 = more obstacles [slightly lower success rate]): ";
            std::cin >> difficulty;
            FAILDETECT
            while(difficulty != 5 && difficulty != 3 && difficulty != 2)
            {
                std::cout << "Only enter 2, 3, OR 5 for difficulty: ";
                std::cin >> difficulty;
                FAILDETECT
            }
            valid = true;
        }
        catch (std::invalid_argument& e)
        {
            std::cout << e.what() << " Going to the beginning.\n\n";
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
    }
    while(!valid);

    std::cout << "\nEnter Robot Type: ";
    std::getline(std::cin, rType);
    createMatrix(matrix, traceMatrix, rName, rType, difficulty);
    displayMatrix(matrix);
    std::cout << "Before starting the program, the robot will use Actice Robot Modern Scan (ARMS) to view any immediate obstacles that make it impossible/pointless for the robot to navigate.\n";
    scanningAnimation();

    if(scanObstacles(matrix))
    {
        std::cout << "Scan detected obstacles blocking the path to the goal.\n";
        std::cout << "Remodification in Progress.\n";
```

```cpp
        DELAY
        recreationAnimation(animatedMatrix);
    }

    while(scanObstacles(matrix))
    {
        count = 0;
        createMatrix(matrix, traceMatrix, rName, rType, difficulty);
        if(!scanObstacles(matrix))
        {
            DELAY
            break;
        }
    }

    std::cout << "No further immediate problems detected. Proceeding with the Program.\n\n";
    DELAY
    DELAY
    matrix[0][0] = Robot(rName, rType);
    clearScreen();
    displayMatrix(matrix);
    DELAY
    matrix[0][0] = Robot(' ', "");
    secondRound(matrix);
    clearTrace(traceMatrix);
    RESETVAR // Reset all the previously declared global variables

    if(noPath)
    {
        count = 2;
    }
    else
    {
        if(needMoreRound)
        {
            count = 1;
        }
        else
        {
            count = 0;
        }
    }

    matrix[4][4] = Robot('*', "*");
    RESETSIM

    if (count == 2)
    {
        std::cout << "\nRare case where the scan was unsuccessful in mitigating the immediate obstacles. Something went wrong. Try again later.\n\n";
    }
    else
    {
        while (!end)
        {
            clearScreen();
            moveCursorUp(11);
            updateMatrix(matrix);
            if (end)
            {
                matrix[row][column] = Robot(' ', "");
                break;
            }
            displayMatrix(matrix);
            loadingDelay();
            //clearScreen();
        }

        int missed = checkMissedPath(matrix);

        if (missed == 0)
        {
            missed++;
        }

        std::cout << "[" << rName << "], which is a(n) [" << rType << "] type robot was successfully able to reach the goal using obstacle avoidance.\n\n";
        SIMSTATS
    }
}

// Create matrix with obstacles and goal
void createMatrix(std::vector<std::vector<Robot>>& matrix, std::vector<std::vector<Robot>>& traceMatrix, char rName, std::string rType, int difficulty)
{
    for (int i = 0; i < matrix.size(); i++)
    {
        for (int j = 0; j < matrix[0].size(); j++)
        {
            if (i == 0 && j == 0)
            {
                matrix[i][j] = Robot(rName, rType);
                traceMatrix[i][j] = Robot('.', "3");
            }
            else if (i == matrix.size() - 1 && j == matrix.size() - 1)
            {
                matrix[i][j] = Robot('*', "*");
                traceMatrix[i][j] = Robot('.', "1");
            }
            else
            {
                int random = rand() % difficulty; // user-preference -> 2, 3, 5
                if (random == 0)
                {
                    matrix[i][j] = Robot('0', "0");
                    traceMatrix[i][j] = Robot('0', "0");
                }
                else
                {
                    matrix[i][j] = Robot(' ', "");
                    traceMatrix[i][j] = Robot(' ', "");
                }
            }
        }
    }
}

// Display initial matrix
void displayMatrix(std::vector<std::vector<Robot>>& matrix)
{
    for (int a = 0; a < matrix.size(); a++)
    {
        std::cout << "_________________________________________________________\n\n";
        for (int b = 0; b < matrix[0].size(); b++)
        {
            if (matrix[a][b].getName() == rName)
            {
                std::cout << "|   \033[1;31m" << matrix[a][b].getName() << "\033[0m   |   ";
            }
            else if(a == 4 && b == 4)
            {
                std::cout << "|   \033[1;33m" << matrix[a][b].getName() << "\033[0m   |   ";
            }
            else
            {
                std::cout << "|   \033[1;32m" << matrix[a][b].getName() << "\033[0m   |   ";
            }
        }
        std::cout << std::endl;
    }
    std::cout << "_________________________________________________________\n\n";
}

// Update robot position in the matrix
void updateMatrix(std::vector<std::vector<Robot>>& matrix)
{
    moves++;
    if(secondStep1)
    {
        matrix[row][column] = Robot(' ', "");
        column++;
        rSensor++;
        matrix[row][column] = Robot(rName, rType);
        traceMatrix[row][column] = Robot('.', "1");
        secondStep1 = false;
        fourDVUse++;
        return;
    }
    if(secondStep2)
    {
        matrix[row][column] = Robot(' ', "");
        row++;
        dSensor++;
        matrix[row][column] = Robot(rName, rType);
        traceMatrix[row][column] = Robot('.', "2");
        secondStep2 = false;
        fourDVUse++;
        return;
    }

    if(count == 0)
    {
        if ((row == 4 && column + 1 < matrix[0].size()) && (matrix[row][column + 1].getName() == '0' && matrix[row - 1][column].getName() != '0' && matrix[row - 1][column + 1].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            row--;
            uSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
            secondStep1 = true;
        }
        else if ((column + 1 < matrix[0].size() && row + 1 < matrix.size() && row - 1 >= 0) && (matrix[row + 1][column].getName() == '0' && matrix[row][column + 1].getName() == '0') && (matrix[row - 1][column + 1].getName() != '0' && matrix[row - 1][column].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            row--;
            uSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
            secondStep1 = true;
        }
        else if ((row == 0 && column == 0) && (matrix[row + 1][column].getName() != '0' && matrix[row + 2][column].getName() == '0') && (matrix[row][column + 1].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            column++;
            rSensor++;
            matrix[row][column] = Robot(rName, rType);
```

```cpp
            traceMatrix[row][column] = Robot('.', "1");
        }
        else if (row + 1 < matrix.size() && matrix[row + 1][column].getName() != '0')
        {
            matrix[row][column] = Robot(' ', "");
            row++;
            dSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
        }
        else if (column + 1 < matrix[0].size() && matrix[row][column + 1].getName() != '0')
        {
            matrix[row][column] = Robot(' ', "");
            column++;
            rSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
        }
        else
        {
            end = true;
        }
    }
    else
    {
        if ((column == 4 && row + 1 < matrix.size()) && (matrix[row + 1][column].getName() == '0' && matrix[row][column - 1].getName() != '0' && matrix[row + 1][column - 1].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            column--;
            lSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
            secondStep2 = true;
        }
        else if ((row + 1 < matrix.size() && column + 1 < matrix[0].size() && column - 1 >= 0) && (matrix[row][column + 1].getName() == '0' && matrix[row + 1][column].getName() == '0') && (matrix[row + 1][column - 1].getName() != '0' && matrix[row][column - 1].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            column--;
            lSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
            secondStep2 = true;
        }
        else if ((column == 0 && row == 0) && (matrix[row][column + 1].getName() != '0' && matrix[row][column + 2].getName() == '0') && (matrix[row + 1][column].getName() != '0'))
        {
            matrix[row][column] = Robot(' ', "");
            row++;
            dSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
        }
        else if (column + 1 < matrix[0].size() && matrix[row][column + 1].getName() != '0')
        {
            matrix[row][column] = Robot(' ', "");
            column++;
            rSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
        }
        else if (row + 1 < matrix.size() && matrix[row + 1][column].getName() != '0')
        {
            matrix[row][column] = Robot(' ', "");
            row++;
            dSensor++;
            matrix[row][column] = Robot(rName, rType);
            traceMatrix[row][column] = Robot('.', "1");
        }
        else
        {
            end = true;
        }
    }
}

// Displays the animated matrix or trace matrix
void showGeneralMatrix(std::vector<std::vector<Robot>>& generalMatrix)
{
    for (int a2 = 0; a2 < generalMatrix.size(); a2++)
    {
        std::cout << "_________________________________________________________\n\n";
        for (int b2 = 0; b2 < generalMatrix[0].size(); b2++)
        {
            if (generalMatrix[a2][b2].getName() != '0')
            {
                if(a2 == 0 && b2 == 0)
                {
                    std::cout << "|   \033[1;31m" << generalMatrix[a2][b2].getName() << "\033[0m   |   ";
                }
                else
                {
                    std::cout << "|   \033[1;34m" << generalMatrix[a2][b2].getName() << "\033[0m   |   ";
                }
            }
            else
            {
                std::cout << "|   " << generalMatrix[a2][b2].getName() << "   |   ";
            }
        }
        std::cout << std::endl;
    }
    std::cout << "_________________________________________________________\n\n";
}

// Loading animation for visual effect
void loadingDelay()
{
    std::this_thread::sleep_for(std::chrono::milliseconds(300));
}

// Scans the grid to check whether or not there are strange obstacles that is obstructing the flow of the program
bool scanObstacles(std::vector<std::vector<Robot>>& matrix)
{
    RESETSIM
    if((matrix[0][1].getName() == '0' && matrix[1][0].getName() == '0') || (matrix[3][4].getName() == '0' && matrix[4][3].getName() == '0'))
    {
        return true;
    }
    count = 0;
    end = false;
    while(!end)
    {
        updateMatrix(matrix);
        if(row == 4 && column == 4)
        {
            matrix[row][column] = Robot(' ', "");
            return false;
        }
        else if(end)
        {
            matrix[row][column] = Robot(' ', "");
            break;
        }
    }
    RESETSIM
    count = 1;
    while(!end)
    {
        updateMatrix(matrix);
        if(row == 4 && column == 4)
        {
            matrix[row][column] = Robot(' ', "");
            return false;
```

```cpp
        }
        else if(end)
        {
            matrix[row][column] = Robot(' ', "");
            break;
        }
    }
    return true;
}

// Checks whether or not it needs a second round -> allows program to display the best round
void secondRound(std::vector<std::vector<Robot>>& matrix)
{
    count = 0;
    RESETSIM

    while (!end)
    {
        updateMatrix(matrix);
        if (row == 4 && column == 4)
        {
            needMoreRound = false;
            matrix[row][column] = Robot(' ', "");
            return;
        }
        else if (end)
        {
            matrix[row][column] = Robot(' ', "");
            break;
        }
    }
    RESETSIM;
    count = 1;
    while (!end)
    {
        updateMatrix(matrix);
        if (row == 4 && column == 4)
        {
            needMoreRound = true;
            matrix[row][column] = Robot(' ', "");
            return;
        }
        else if (end)
        {
            noPath = true;
            matrix[row][column] = Robot(' ', "");
            break;
        }
    }
}
```

```cpp

// Scanning animation for visual effect
void scanningAnimation()
{
    for(int y = 0; y < 3; y++)
    {
        if(y == 0)
        {
            std::cout << "Scanning.  ";
        }
        else if(y == 1)
        {
            std::cout << "Scanning.. ";
        }
        else
        {
            std::cout << "Scanning...\n\n";
        }
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
}

// Recreating animation for visual effect
void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix)
{
    moveCursorUp(11);
    std::this_thread::sleep_for(std::chrono::milliseconds(350));
    animatedMatrix[0][0] = Robot(rName, rType);
    animatedMatrix[4][4] = Robot('*', "*");
    clearScreen();
    if(repeat == 0)
    {
        ANIMATE1
    }
    else if(repeat == 1)
    {
        ANIMATE2
    }
    else if(repeat == 2)
    {
        ANIMATE3
    }
    else if(repeat == 3)
    {
        ANIMATE4
    }
    else if(repeat == 4)
    {
        ANIMATE5
    }
    else if(repeat == 5)
    {
        ANIMATE6
    }
    else
    {
        ANIMATE7
    }

    CLEARANIMATION

    if(repeat <= 6)
    {
        recreationAnimation(animatedMatrix); // Recrusive call
    }
}

// Checks to see the total number of paths to goal (if there are any)
int checkMissedPath(std::vector<std::vector<Robot>>& matrix)
{
    int c = 0;
    int num = checkPerimeterPath(matrix);
    for(int r = 3; r >= 1; r--)
    {
        c++;
        if(matrix[r][c].getName() != '0' && (matrix[r + 1][c].getName() != '0' || matrix[r][c + 1].getName() != '0'))
        {
            if(matrix[r + 1][c + 1].getName() != '0')
            {
                if((matrix[r - 1][c].getName() != '0' || matrix[r][c - 1].getName() != '0') && (matrix[1][2].getName() != '0' || matrix[2][1].getName() != '0'))
                {
                    if(r == 3)
                    {
                        if(matrix[4][3].getName() != '0')
                        {
                            num++;
                        }
                    }
                    else
                    {
                        num++;
                    }
                }
            }
        }
    }
    return num;
}
```

```cpp

// Checks the perimeter of the grid to see if there are any pathways there that could have been taken
int checkPerimeterPath(std::vector<std::vector<Robot>>& matrix)
{
    int z = 0;
    int l = 0;
    int l2 = 4;
    int z2 = 4;
    int result = 0;

    for(int u = 0; u < 2; u++)
    {
        l = 0;
        if(u == 0)
        {
            while(l < 5)
            {
                if(matrix[z][l].getName() == '0' || matrix[l][l2].getName() == '0')
                {
                    break;
                }
                l++;
            }

            if(l == 5)
            {
                result++;
            }
        }
        else
        {
            while(z < 5)
            {
                if(matrix[z][l].getName() == '0' || matrix[z2][z].getName() == '0')
                {
                    break;
                }
                z++;
            }

            if(z == 5)
            {
                result++;
            }
        }
    }
    return result;
}

```

```cpp
// Clears any unintended footprints in trace matrix before real display
void clearTrace(std::vector<std::vector<Robot>>& traceMatrix)
{
    for(int q1 = 0; q1 < traceMatrix.size(); q1++)
    {
        for(int w1 = (q1 == 0) ? w1 = 1 : w1 = 0; w1 < traceMatrix[0].size(); w1++)
        {
            if(traceMatrix[q1][w1].getName() != '0')
            {
                traceMatrix[q1][w1] = Robot(' ', "");
            }
        }
    }
}

// ANSI escape sequences for terminal cursor control
void clearScreen()
{
    std::cout << "\033[2J\033[H";
}

void moveCursorUp(int lines)
{
    std::cout << "\033[" << lines << "A";
}
```