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;
```

```
32 bool end = false;
33 int count = 0;
34 int moves = 0;
35 bool secondStep1 = false;
36 bool secondStep2 = false;
37 int repeat = 0;
38 int fourDVUse = 0;
39 int rSensor = 0;
40 int dSensor = 0;
41 int uSensor = 0;
42 int lSensor = 0;
43 int difficulty;
44 bool needMoreRound = true;
45 bool noPath = false;
46
47 // Function Prototypes
48 void createMatrix(std::vector<std::vector<Robot>>& matrix, std::vector<std::vector<Robot>>&
   traceMatrix, char rName, std::string rType, int difficulty);
49 void displayMatrix(std::vector<std::vector<Robot>>& matrix);
50 void updateMatrix(std::vector<std::vector<Robot>>& matrix);
51 bool scanObstacles(std::vector<std::vector<Robot>>& matrix);
52 void secondRound(std::vector<std::vector<Robot>>& matrix);
53 void loadingDelay();
54 void scanningAnimation();
55 void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix);
56 void showGeneralMatrix(std::vector<std::vector<Robot>>& generalMatrix);
57 int checkMissedPath(std::vector<std::vector<Robot>>& matrix);
58 int checkPerimeterPath(std::vector<std::vector<Robot>>& matrix);
59 void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix);
60 void clearTrace(std::vector<std::vector<Robot>>& traceMatrix);
61 void moveCursorUp(int lines);
62 void clearScreen();
63
64 // Runs all the main functions and text
65 int main()
66 {
67     srand(time(NULL));
68     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";
69     std::cout << "Simulation Success Rate (Tested 30 Times): 98%\n\n";
70     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";
71     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";
72     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";
73     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";
74      bool valid = false;
75      do
76      {
77          try
78          {
79              std::cout << "Enter Robot Name (Single Character. Cannot be '0' or 'O'): ";
80              std::cin >> rName;
81              FAILDETECT
82              while(rName == '0' || rName == 'O')
83              {
84                  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': ";
85                  std::cin >> rName;
86              FAILDETECT
87              }
88              std::cout << "\nEnter difficulty (5 = less obstacles [super high success rate], 3 =
mild obstacles [good success rate], 2 = more obstacles [slightly lower success rate]): ";
89              std::cin >> difficulty;
90              FAILDETECT
91              while(difficulty != 5 && difficulty != 3 && difficulty != 2)
92              {
93                  std::cout << "Only enter 2, 3, OR 5 for difficulty: ";
94                  std::cin >> difficulty;
95                  FAILDETECT
96              }
97              valid = true;
98          }
99          catch (std::invalid_argument& e)
100         {
101             std::cout << e.what() << " Going to the beginning.\n\n";
102             std::cin.clear();
103             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
104         }
105     }
106     while(!valid);
107
108     std::cout << "\nEnter Robot Type: ";
109     std::getline(std::cin, rType);
110     createMatrix(matrix, traceMatrix, rName, rType, difficulty);
111     displayMatrix(matrix);
112     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";
113     scanningAnimation();
114
115     if(scanObstacles(matrix))
116     {
117         std::cout << "Scan detected obstacles blocking the path to the goal.\n";
118         std::cout << "Remodification in Progress.\n";
```

```
119         DELAY
120         recreationAnimation(animatedMatrix);
121     }
122 
123     while(scanObstacles(matrix))
124     {
125         count = 0;
126         createMatrix(matrix, traceMatrix, rName, rType, difficulty);
127         if(!scanObstacles(matrix))
128         {
129             DELAY
130             break;
131         }
132     }
133 
134     std::cout << "No further immediate problems detected. Proceeding with the Program.\n\n";
135     DELAY
136     DELAY
137     matrix[0][0] = Robot(rName, rType);
138     clearScreen();
139     displayMatrix(matrix);
140     DELAY
141     matrix[0][0] = Robot(' ', "");
142     secondRound(matrix);
143     clearTrace(traceMatrix);
144     RESETVAR // Reset all the previously declared global variables
145 
146     if(noPath)
147     {
148         count = 2;
149     }
150     else
151     {
152         if(needMoreRound)
153         {
154             count = 1;
155         }
156         else
157         {
158             count = 0;
159         }
160     }
161 
162     matrix[4][4] = Robot('*', "*");
163     RESETSIM
164 
165     if (count == 2)
166     {
167         std::cout << "\nRare case where the scan was unsuccessful in mitigating the immediate
    obstacles. Something went wrong. Try again later.\n\n";
```

```
168     }
169     else
170     {
171         while (!end)
172         {
173             clearScreen();
174             moveCursorUp(11);
175             updateMatrix(matrix);
176             if (end)
177             {
178                 matrix[row][column] = Robot(' ', "");
179                 break;
180             }
181             displayMatrix(matrix);
182             loadingDelay();
183             //clearScreen();
184         }
185 
186         int missed = checkMissedPath(matrix);
187 
188         if (missed == 0)
189         {
190             missed++;
191         }
192 
193         std::cout << "[" << rName << "], which is a(n) [" << rType << "] type robot was
    successfully able to reach the goal using obstacle avoidance.\n\n";
194         SIMSTATS
195     }
196 }
197 
198 // Create matrix with obstacles and goal
199 void createMatrix(std::vector<std::vector<Robot>>& matrix, std::vector<std::vector<Robot>>&
    traceMatrix, char rName, std::string rType, int difficulty)
200 {
201     for (int i = 0; i < matrix.size(); i++)
202     {
203         for (int j = 0; j < matrix[0].size(); j++)
204         {
205             if (i == 0 && j == 0)
206             {
207                 matrix[i][j] = Robot(rName, rType);
208                 traceMatrix[i][j] = Robot('.', "3");
209             }
210             else if (i == matrix.size() - 1 && j == matrix.size() - 1)
211             {
212                 matrix[i][j] = Robot('*', "*");
213                 traceMatrix[i][j] = Robot('.', "1");
214             }
215             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(' ', "");
```

```
266         column++;
267         rSensor++;
268         matrix[row][column] = Robot(rName, rType);
269         traceMatrix[row][column] = Robot('.', "1");
270         secondStep1 = false;
271         fourDVUse++;
272         return;
273     }
274     if(secondStep2)
275     {
276         matrix[row][column] = Robot(' ', "");
277         row++;
278         dSensor++;
279         matrix[row][column] = Robot(rName, rType);
280         traceMatrix[row][column] = Robot('.', "2");
281         secondStep2 = false;
282         fourDVUse++;
283         return;
284     }
285 
286     if(count == 0)
287     {
288         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'))
289         {
290             matrix[row][column] = Robot(' ', "");
291             row--;
292             uSensor++;
293             matrix[row][column] = Robot(rName, rType);
294             traceMatrix[row][column] = Robot('.', "1");
295             secondStep1 = true;
296         }
297         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'))
298         {
299             matrix[row][column] = Robot(' ', "");
300             row--;
301             uSensor++;
302             matrix[row][column] = Robot(rName, rType);
303             traceMatrix[row][column] = Robot('.', "1");
304             secondStep1 = true;
305         }
306         else if ((row == 0 && column == 0) && (matrix[row + 1][column].getName() != '0' &&
    matrix[row + 2][column].getName() == '0') && (matrix[row][column + 1].getName() != '0'))
307         {
308             matrix[row][column] = Robot(' ', "");
309             column++;
310             rSensor++;
311             matrix[row][column] = Robot(rName, rType);
```

```
312             traceMatrix[row][column] = Robot('.', "1");
313         }
314         else if (row + 1 < matrix.size() && matrix[row + 1][column].getName() != '0')
315         {
316             matrix[row][column] = Robot(' ', "");
317             row++;
318             dSensor++;
319             matrix[row][column] = Robot(rName, rType);
320             traceMatrix[row][column] = Robot('.', "1");
321         }
322         else if (column + 1 < matrix[0].size() && matrix[row][column + 1].getName() != '0')
323         {
324             matrix[row][column] = Robot(' ', "");
325             column++;
326             rSensor++;
327             matrix[row][column] = Robot(rName, rType);
328             traceMatrix[row][column] = Robot('.', "1");
329         }
330         else
331         {
332             end = true;
333         }
334     }
335     else
336     {
337         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'))
338         {
339             matrix[row][column] = Robot(' ', "");
340             column--;
341             lSensor++;
342             matrix[row][column] = Robot(rName, rType);
343             traceMatrix[row][column] = Robot('.', "1");
344             secondStep2 = true;
345         }
346         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'))
347         {
348             matrix[row][column] = Robot(' ', "");
349             column--;
350             lSensor++;
351             matrix[row][column] = Robot(rName, rType);
352             traceMatrix[row][column] = Robot('.', "1");
353             secondStep2 = true;
354         }
355         else if ((column == 0 && row == 0) && (matrix[row][column + 1].getName() != '0' &&
    matrix[row][column + 2].getName() == '0') && (matrix[row + 1][column].getName() != '0'))
356         {
357             matrix[row][column] = Robot(' ', "");
```

```
358             row++;
359             dSensor++;
360             matrix[row][column] = Robot(rName, rType);
361             traceMatrix[row][column] = Robot('.', "1");
362         }
363         else if (column + 1 < matrix[0].size() && matrix[row][column + 1].getName() != '0')
364         {
365             matrix[row][column] = Robot(' ', "");
366             column++;
367             rSensor++;
368             matrix[row][column] = Robot(rName, rType);
369             traceMatrix[row][column] = Robot('.', "1");
370         }
371         else if (row + 1 < matrix.size() && matrix[row + 1][column].getName() != '0')
372         {
373             matrix[row][column] = Robot(' ', "");
374             row++;
375             dSensor++;
376             matrix[row][column] = Robot(rName, rType);
377             traceMatrix[row][column] = Robot('.', "1");
378         }
379         else
380         {
381             end = true;
382         }
383     }
384 }
385
386 // Displays the animated matrix or trace matrix
387 void showGeneralMatrix(std::vector<std::vector<Robot>>& generalMatrix)
388 {
389     for (int a2 = 0; a2 < generalMatrix.size(); a2++)
390     {
391         std::cout << "_____________________________________________________________\n\n";
392         for (int b2 = 0; b2 < generalMatrix[0].size(); b2++)
393         {
394             if (generalMatrix[a2][b2].getName() != '0')
395             {
396                 if(a2 == 0 && b2 == 0)
397                 {
398                     std::cout << "|   \033[1;31m" << generalMatrix[a2][b2].getName() <<
    "\033[0m   |   ";
399                 }
400                 else
401                 {
402                     std::cout << "|   \033[1;34m" << generalMatrix[a2][b2].getName() <<
    "\033[0m   |   ";
403                 }
404             }
405             else
```

```
406             {
407                 std::cout << "|   " << generalMatrix[a2][b2].getName() << "   |   ";
408             }
409         }
410         std::cout << std::endl;
411     }
412     std::cout << "_____________________________________________________________\n\n";
413 }
414 
415 // Loading animation for visual effect
416 void loadingDelay()
417 {
418     std::this_thread::sleep_for(std::chrono::milliseconds(300));
419 }
420 
421 // Scans the grid to check whether or not there are strange obstacles that is obstructing the
    flow of the program
422 bool scanObstacles(std::vector<std::vector<Robot>>& matrix)
423 {
424     RESETSIM
425     if((matrix[0][1].getName() == '0' && matrix[1][0].getName() == '0') || (matrix[3]
    [4].getName() == '0' && matrix[4][3].getName() == '0'))
426     {
427         return true;
428     }
429     count = 0;
430     end = false;
431     while(!end)
432     {
433         updateMatrix(matrix);
434         if(row == 4 && column == 4)
435         {
436             matrix[row][column] = Robot(' ', "");
437             return false;
438         }
439         else if(end)
440         {
441             matrix[row][column] = Robot(' ', "");
442             break;
443         }
444     }
445     RESETSIM
446     count = 1;
447     while(!end)
448     {
449         updateMatrix(matrix);
450         if(row == 4 && column == 4)
451         {
452             matrix[row][column] = Robot(' ', "");
453             return false;
```

```
454         }
455         else if(end)
456         {
457             matrix[row][column] = Robot(' ', "");
458             break;
459         }
460     }
461     return true;
462 }
463 
464 // Checks whether or not it needs a second round -> allows program to display the best round
465 void secondRound(std::vector<std::vector<Robot>>& matrix)
466 {
467     count = 0;
468     RESETSIM
469 
470     while (!end)
471     {
472         updateMatrix(matrix);
473         if (row == 4 && column == 4)
474         {
475             needMoreRound = false;
476             matrix[row][column] = Robot(' ', "");
477             return;
478         }
479         else if (end)
480         {
481             matrix[row][column] = Robot(' ', "");
482             break;
483         }
484     }
485     RESETSIM;
486     count = 1;
487     while (!end)
488     {
489         updateMatrix(matrix);
490         if (row == 4 && column == 4)
491         {
492             needMoreRound = true;
493             matrix[row][column] = Robot(' ', "");
494             return;
495         }
496         else if (end)
497         {
498             noPath = true;
499             matrix[row][column] = Robot(' ', "");
500             break;
501         }
502     }
503 }
```

```
504
505 // Scanning animation for visual effect
506 void scanningAnimation()
507 {
508     for(int y = 0; y < 3; y++)
509     {
510         if(y == 0)
511         {
512             std::cout << "Scanning. ";
513         }
514         else if(y == 1)
515         {
516             std::cout << "Scanning.. ";
517         }
518         else
519         {
520             std::cout << "Scanning...\n\n";
521         }
522         std::this_thread::sleep_for(std::chrono::seconds(1));
523     }
524 }
525
526 // Recreating animation for visual effect
527 void recreationAnimation(std::vector<std::vector<Robot>>& animatedMatrix)
528 {
529     moveCursorUp(11);
530     std::this_thread::sleep_for(std::chrono::milliseconds(350));
531     animatedMatrix[0][0] = Robot(rName, rType);
532     animatedMatrix[4][4] = Robot('*', "*");
533     clearScreen();
534     if(repeat == 0)
535     {
536         ANIMATE1
537     }
538     else if(repeat == 1)
539     {
540         ANIMATE2
541     }
542     else if(repeat == 2)
543     {
544         ANIMATE3
545     }
546     else if(repeat == 3)
547     {
548         ANIMATE4
549     }
550     else if(repeat == 4)
551     {
552         ANIMATE5
553     }
```

```
554     else if(repeat == 5)
555     {
556         ANIMATE6
557     }
558     else
559     {
560         ANIMATE7
561     }
562 
563     CLEARANIMATION
564 
565     if(repeat <= 6)
566     {
567         recreationAnimation(animatedMatrix); // Recrusive call
568     }
569 }
570 
571 // Checks to see the total number of paths to goal (if there are any)
572 int checkMissedPath(std::vector<std::vector<Robot>>& matrix)
573 {
574     int c = 0;
575     int num = checkPerimeterPath(matrix);
576     for(int r = 3; r >= 1; r--)
577     {
578         c++;
579         if(matrix[r][c].getName() != '0' && (matrix[r + 1][c].getName() != '0' || matrix[r][c +
    1].getName() != '0'))
580         {
581             if(matrix[r + 1][c + 1].getName() != '0')
582             {
583                 if((matrix[r - 1][c].getName() != '0' || matrix[r][c - 1].getName() != '0') &&
    (matrix[1][2].getName() != '0' || matrix[2][1].getName() != '0'))
584                 {
585                     if(r == 3)
586                     {
587                         if(matrix[4][3].getName() != '0')
588                         {
589                             num++;
590                         }
591                     }
592                     else
593                     {
594                         num++;
595                     }
596                 }
597             }
598         }
599     }
600     return num;
601 }
```

```
602
603 // Checks the perimeter of the grid to see if there are any pathways there that could have been
    taken
604 int checkPerimeterPath(std::vector<std::vector<Robot>>& matrix)
605 {
606     int z = 0;
607     int l = 0;
608     int l2 = 4;
609     int z2 = 4;
610     int result = 0;
611
612     for(int u = 0; u < 2; u++)
613     {
614         l = 0;
615         if(u == 0)
616         {
617             while(l < 5)
618             {
619                 if(matrix[z][l].getName() == '0' || matrix[l][l2].getName() == '0')
620                 {
621                     break;
622                 }
623                 l++;
624             }
625
626             if(l == 5)
627             {
628                 result++;
629             }
630         }
631         else
632         {
633             while(z < 5)
634             {
635                 if(matrix[z][l].getName() == '0' || matrix[z2][z].getName() == '0')
636                 {
637                     break;
638                 }
639                 z++;
640             }
641
642             if(z == 5)
643             {
644                 result++;
645             }
646         }
647     }
648     return result;
649 }
650
```

```
651 // Clears any unintended footprints in trace matrix before real display
652 void clearTrace(std::vector<std::vector<Robot>>& traceMatrix)
653 {
654     for(int q1 = 0; q1 < traceMatrix.size(); q1++)
655     {
656         for(int w1 = (q1 == 0) ? w1 = 1 : w1 = 0; w1 < traceMatrix[0].size(); w1++)
657         {
658             if(traceMatrix[q1][w1].getName() != '0')
659             {
660                 traceMatrix[q1][w1] = Robot(' ', "");
661             }
662         }
663     }
664 }
665
666 // ANSI escape sequences for terminal cursor control
667 void clearScreen()
668 {
669     std::cout << "\033[2J\033[H";
670 }
671
672 void moveCursorUp(int lines)
673 {
674     std::cout << "\033[" << lines << "A";
675 }
```