|  |
| --- |
| *// A C++ Program to implement A\* Search Algorithm*  #include <iostream>  #include<stack>  #include<vector>  #include<set>  using namespace std;  #define ROW 9  #define COL 10  typedef pair<int, int> Pair;  typedef pair<double, pair<int, int> > pPair;  struct cell {      int parent\_i, parent\_j;      double f, g, h;  };  bool isValid(int *row*, int *col*)  {  *// Returns true if row number and column number is in range*      return (row >= 0) && (row < ROW) && (col >= 0)&& (col < COL);  }  bool isUnBlocked(int *grid*[][COL], int *row*, int *col*)  {  *// Returns true if the cell is not blocked else false*      if (grid[row][col] == 1)          return (true);      else          return (false);  }  bool isDestination(int *row*, int *col*, Pair *dest*)  {      if (row == dest.first && col == dest.second)          return (true);      else          return (false);  }  double calculateHValue(int *row*, int *col*, Pair *dest*)  {  *// Return using the distance formula*  *// h = sqrt ((currrent\_cell.x - goal.x)2+(current\_cell - goal.y)2)*      return ((double)sqrt((row - dest.first) \* (row - dest.first)+ (col - dest.second) \* (col - dest.second)));  }  void tracePath(cell *cellDetails*[][COL], Pair *dest*)  {      printf("\nThe Path is ");      int row = dest.first;      int col = dest.second;      stack<Pair> Path;      while (!(cellDetails[row][col].parent\_i == row && cellDetails[row][col].parent\_j == col)) {          Path.push(make\_pair(row, col));          int temp\_row = cellDetails[row][col].parent\_i;          int temp\_col = cellDetails[row][col].parent\_j;          row = temp\_row;          col = temp\_col;      }      Path.push(make\_pair(row, col));      while (!Path.empty()) {          pair<int, int> p = Path.top();          Path.pop();          printf("-> (%d,%d) ", p.first, p.second);      }      return;  }  void aStarSearch(int *grid*[][COL], Pair *src*, Pair *dest*)  {  *// If the source is out of range*      if (isValid(src.first, src.second) == false) {          printf("Source is invalid\n");          return;      }  *// If the destination is out of range*      if (isValid(dest.first, dest.second) == false) {          printf("Destination is invalid\n");          return;      }  *// Either the source or the destination is blocked*      if (isUnBlocked(grid, src.first, src.second) == false || isUnBlocked(grid, dest.first, dest.second) == false)      {          printf("Source or the destination is blocked\n");          return;      }  *// If the destination cell is the same as source cell*      if (isDestination(src.first, src.second, dest)          == true) {          printf("We are already at the destination\n");          return;      }  *// Create a closed list and initialise it to false which means that no cell has been included yet This closed list is implemented as a boolean 2D array*      bool closedList[ROW][COL];      memset(closedList, false, sizeof(closedList));  *// Declare a 2D array of structure to hold the details of that cell*      cell cellDetails[ROW][COL];      int i, j;      for (i = 0; i < ROW; i++) {          for (j = 0; j < COL; j++) {              cellDetails[i][j].f = FLT\_MAX;              cellDetails[i][j].g = FLT\_MAX;              cellDetails[i][j].h = FLT\_MAX;              cellDetails[i][j].parent\_i = -1;              cellDetails[i][j].parent\_j = -1;          }      }  *// Initialising the parameters of the starting node*      i = src.first, j = src.second;      cellDetails[i][j].f = 0.0;      cellDetails[i][j].g = 0.0;      cellDetails[i][j].h = 0.0;      cellDetails[i][j].parent\_i = i;      cellDetails[i][j].parent\_j = j;      set<pPair> openList;  *// Put the starting cell on the open list and set its 'f' as 0*      openList.insert(make\_pair(0.0, make\_pair(i, j)));  *// We set this boolean value as false as initially the destination is not reached.*      bool foundDest = false;      while (!openList.empty()) {          pPair p = \*openList.begin();          openList.erase(openList.begin());          i = p.second.first;          j = p.second.second;          closedList[i][j] = true;          double gNew, hNew, fNew;          if (isValid(i - 1, j) == true) {              if (isDestination(i - 1, j, dest) == true) {                  cellDetails[i - 1][j].parent\_i = i;                  cellDetails[i - 1][j].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i - 1][j] == false && isUnBlocked(grid, i - 1, j)== true) {                  gNew = cellDetails[i][j].g + 1.0;                  hNew = calculateHValue(i - 1, j, dest);                  fNew = gNew + hNew;                  if (cellDetails[i - 1][j].f == FLT\_MAX || cellDetails[i - 1][j].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i - 1, j)));                      cellDetails[i - 1][j].f = fNew;                      cellDetails[i - 1][j].g = gNew;                      cellDetails[i - 1][j].h = hNew;                      cellDetails[i - 1][j].parent\_i = i;                      cellDetails[i - 1][j].parent\_j = j;                  }              }          }  *// Only process this cell if this is a valid one*          if (isValid(i + 1, j) == true) {  *// If the destination cell is the same as the current successor*              if (isDestination(i + 1, j, dest) == true) {  *// Set the Parent of the destination cell*                  cellDetails[i + 1][j].parent\_i = i;                  cellDetails[i + 1][j].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i + 1][j] == false&& isUnBlocked(grid, i + 1, j) == true) {                  gNew = cellDetails[i][j].g + 1.0;                  hNew = calculateHValue(i + 1, j, dest);                  fNew = gNew + hNew;                  if (cellDetails[i + 1][j].f == FLT\_MAX || cellDetails[i + 1][j].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i + 1, j)));  *// Update the details of this cell*                      cellDetails[i + 1][j].f = fNew;                      cellDetails[i + 1][j].g = gNew;                      cellDetails[i + 1][j].h = hNew;                      cellDetails[i + 1][j].parent\_i = i;                      cellDetails[i + 1][j].parent\_j = j;                  }              }          }          if (isValid(i, j + 1) == true) {              if (isDestination(i, j + 1, dest) == true) {                  cellDetails[i][j + 1].parent\_i = i;                  cellDetails[i][j + 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i][j + 1] == false && isUnBlocked(grid, i, j + 1) == true) {                  gNew = cellDetails[i][j].g + 1.0;                  hNew = calculateHValue(i, j + 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i][j + 1].f == FLT\_MAX || cellDetails[i][j + 1].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i, j + 1)));  *// Update the details of this cell*                      cellDetails[i][j + 1].f = fNew;                      cellDetails[i][j + 1].g = gNew;                      cellDetails[i][j + 1].h = hNew;                      cellDetails[i][j + 1].parent\_i = i;                      cellDetails[i][j + 1].parent\_j = j;                  }              }          }          if (isValid(i, j - 1) == true) {              if (isDestination(i, j - 1, dest) == true) {                  cellDetails[i][j - 1].parent\_i = i;                  cellDetails[i][j - 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i][j - 1] == false&& isUnBlocked(grid, i, j - 1) == true) {                  gNew = cellDetails[i][j].g + 1.0;                  hNew = calculateHValue(i, j - 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i][j - 1].f == FLT\_MAX || cellDetails[i][j - 1].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i, j - 1)));                      cellDetails[i][j - 1].f = fNew;                      cellDetails[i][j - 1].g = gNew;                      cellDetails[i][j - 1].h = hNew;                      cellDetails[i][j - 1].parent\_i = i;                      cellDetails[i][j - 1].parent\_j = j;                  }              }          }          if (isValid(i - 1, j + 1) == true) {              if (isDestination(i - 1, j + 1, dest) == true) {  *// Set the Parent of the destination cell*                  cellDetails[i - 1][j + 1].parent\_i = i;                  cellDetails[i - 1][j + 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i - 1][j + 1] == false&& isUnBlocked(grid, i - 1, j + 1) == true) {                  gNew = cellDetails[i][j].g + 1.414;                  hNew = calculateHValue(i - 1, j + 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i - 1][j + 1].f == FLT\_MAX                      || cellDetails[i - 1][j + 1].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i - 1, j + 1)));                      cellDetails[i - 1][j + 1].f = fNew;                      cellDetails[i - 1][j + 1].g = gNew;                      cellDetails[i - 1][j + 1].h = hNew;                      cellDetails[i - 1][j + 1].parent\_i = i;                      cellDetails[i - 1][j + 1].parent\_j = j;                  }              }          }          if (isValid(i - 1, j - 1) == true) {              if (isDestination(i - 1, j - 1, dest) == true) {  *// Set the Parent of the destination cell*                  cellDetails[i - 1][j - 1].parent\_i = i;                  cellDetails[i - 1][j - 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i - 1][j - 1] == false                      && isUnBlocked(grid, i - 1, j - 1)                              == true) {                  gNew = cellDetails[i][j].g + 1.414;                  hNew = calculateHValue(i - 1, j - 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i - 1][j - 1].f == FLT\_MAX                      || cellDetails[i - 1][j - 1].f > fNew) {                      openList.insert(make\_pair(                          fNew, make\_pair(i - 1, j - 1)));  *// Update the details of this cell*                      cellDetails[i - 1][j - 1].f = fNew;                      cellDetails[i - 1][j - 1].g = gNew;                      cellDetails[i - 1][j - 1].h = hNew;                      cellDetails[i - 1][j - 1].parent\_i = i;                      cellDetails[i - 1][j - 1].parent\_j = j;                  }              }          }          if (isValid(i + 1, j + 1) == true) {              if (isDestination(i + 1, j + 1, dest) == true) {                  cellDetails[i + 1][j + 1].parent\_i = i;                  cellDetails[i + 1][j + 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i + 1][j + 1] == false                      && isUnBlocked(grid, i + 1, j + 1)                              == true) {                  gNew = cellDetails[i][j].g + 1.414;                  hNew = calculateHValue(i + 1, j + 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i + 1][j + 1].f == FLT\_MAX || cellDetails[i + 1][j + 1].f > fNew) {                      openList.insert(make\_pair(fNew, make\_pair(i + 1, j + 1)));                      cellDetails[i + 1][j + 1].f = fNew;                      cellDetails[i + 1][j + 1].g = gNew;                      cellDetails[i + 1][j + 1].h = hNew;                      cellDetails[i + 1][j + 1].parent\_i = i;                      cellDetails[i + 1][j + 1].parent\_j = j;                  }              }          }          if (isValid(i + 1, j - 1) == true) {              if (isDestination(i + 1, j - 1, dest) == true) {                  cellDetails[i + 1][j - 1].parent\_i = i;                  cellDetails[i + 1][j - 1].parent\_j = j;                  printf("The destination cell is found\n");                  tracePath(cellDetails, dest);                  foundDest = true;                  return;              }              else if (closedList[i + 1][j - 1] == false                      && isUnBlocked(grid, i + 1, j - 1)                              == true) {                  gNew = cellDetails[i][j].g + 1.414;                  hNew = calculateHValue(i + 1, j - 1, dest);                  fNew = gNew + hNew;                  if (cellDetails[i + 1][j - 1].f == FLT\_MAX                      || cellDetails[i + 1][j - 1].f > fNew) {                      openList.insert(make\_pair(                          fNew, make\_pair(i + 1, j - 1)));                      cellDetails[i + 1][j - 1].f = fNew;                      cellDetails[i + 1][j - 1].g = gNew;                      cellDetails[i + 1][j - 1].h = hNew;                      cellDetails[i + 1][j - 1].parent\_i = i;                      cellDetails[i + 1][j - 1].parent\_j = j;                  }              }          }      }      if (foundDest == false)          printf("Failed to find the Destination Cell\n");      return;  }  int main()  {      int grid[ROW][COL]          = { { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },              { 1, 1, 1, 0, 1, 1, 1, 0, 1, 1 },              { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1 },              { 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 },              { 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },              { 1, 0, 1, 1, 1, 1, 0, 1, 0, 0 },              { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 },              { 1, 0, 1, 1, 1, 1, 0, 1, 1, 1 },              { 1, 1, 1, 0, 0, 0, 1, 0, 0, 1 } };      Pair src = make\_pair(0, 9);      Pair dest = make\_pair(0, 0);      aStarSearch(grid, src, dest);      return (0);  } |

A screenshot of a computer

Description automatically generated with medium confidence