AI Tutorial 5

Write a program to Implement A* Algorithm.

Code:-

```cpp
#include <list>

#include <algorithm>

#include <iostream>


class point {

public:

    point( int a = 0, int b = 0 ) { x = a; y = b; }

    bool operator ==( const point& o ) { return o.x == x && o.y == y; }

    point operator +( const point& o ) { return point( o.x + x, o.y + y ); }

    int x, y;

};


class map {

public:

    map() {

        char t[8][8] = {

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

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

        };

        w = h = 8;

        for( int r = 0; r < h; r++ )

            for( int s = 0; s < w; s++ )
```

```cpp
            m[s][r] = t[r][s];
    }
    int operator() ( int x, int y ) { return m[x][y]; }
    char m[8][8];
    int w, h;
};


class node {
public:
    bool operator == (const node& o ) { return pos == o.pos; }
    bool operator == (const point& o ) { return pos == o; }
    bool operator < (const node& o ) { return dist + cost < o.dist + o.cost; }
    point pos, parent;
    int dist, cost;
};


class aStar {
public:
    aStar() {
        neighbours[0] = point( -1, -1 ); neighbours[1] = point(  1, -1 );
        neighbours[2] = point( -1,  1 ); neighbours[3] = point(  1,  1 );
        neighbours[4] = point(  0, -1 ); neighbours[5] = point( -1,  0 );
        neighbours[6] = point(  0,  1 ); neighbours[7] = point(  1,  0 );
    }


    int calcDist( point& p ){
        // need a better heuristic
        int x = end.x - p.x, y = end.y - p.y;
        return( x * x + y * y );
    }
```

```cpp
bool isValid( point& p ) {
    return ( p.x >-1 && p.y > -1 && p.x < m.w && p.y < m.h );
}


bool existPoint( point& p, int cost ) {
    std::list<node>::iterator i;
    i = std::find( closed.begin(), closed.end(), p );
    if( i != closed.end() ) {
        if( ( *i ).cost + ( *i ).dist < cost ) return true;
        else { closed.erase( i ); return false; }
    }
    i = std::find( open.begin(), open.end(), p );
    if( i != open.end() ) {
        if( ( *i ).cost + ( *i ).dist < cost ) return true;
        else { open.erase( i ); return false; }
    }
    return false;
}


bool fillOpen( node& n ) {
    int stepCost, nc, dist;
    point neighbour;


    for( int x = 0; x < 8; x++ ) {
        // one can make diagonals have different cost
        stepCost = x < 4 ? 1 : 1;
        neighbour = n.pos + neighbours[x];
        if( neighbour == end ) return true;
```

```cpp
        if( isValid( neighbour ) && m( neighbour.x, neighbour.y ) != 1 ) {

          nc = stepCost + n.cost;

          dist = calcDist( neighbour );

          if( !existPoint( neighbour, nc + dist ) ) {

            node m;

            m.cost = nc; m.dist = dist;

            m.pos = neighbour;

            m.parent = n.pos;

            open.push_back( m );

          }

        }

    }

    return false;

}


bool search( point& s, point& e, map& mp ) {

    node n; end = e; start = s; m = mp;

    n.cost = 0; n.pos = s; n.parent = 0; n.dist = calcDist( s );

    open.push_back( n );

    while( !open.empty() ) {

      //open.sort();

      node n = open.front();

      open.pop_front();

      closed.push_back( n );

      if( fillOpen( n ) ) return true;

    }

    return false;

}


int path( std::list<point>& path ) {
```

```cpp
        path.push_front( end );

        int cost = 1 + closed.back().cost;

        path.push_front( closed.back().pos );

        point parent = closed.back().parent;


        for( std::list<node>::reverse_iterator i = closed.rbegin(); i != closed.rend(); i++ ) {

            if( ( *i ).pos == parent && !( ( *i ).pos == start ) ) {

                path.push_front( ( *i ).pos );

                parent = ( *i ).parent;

            }

        }

        path.push_front( start );

        return cost;

    }


    map m; point end, start;

    point neighbours[8];

    std::list<node> open;

    std::list<node> closed;

};

int main( int argc, char* argv[] ) {

    map m;

    point s, e( 7, 7 );

    aStar as;


    if( as.search( s, e, m ) ) {

        std::list<point> path;

        int c = as.path( path );

        for( int y = -1; y < 9; y++ ) {
```

```
        for( int x = -1; x < 9; x++ ) {

          if( x < 0 || y < 0 || x > 7 || y > 7 || m( x, y ) == 1 )

            std::cout << char(0xdb);

          else {

            if( std::find( path.begin(), path.end(), point( x, y ) )!= path.end() )

              std::cout << "x";

            else std::cout << ".";

          }

        }

        std::cout << "\n";

      }


      std::cout << "\nPath cost " << c << ": ";

      for( std::list<point>::iterator i = path.begin(); i != path.end(); i++ ) {

        std::cout<< "(" << ( *i ).x << ", " << ( *i ).y << ") ";

      }

    }

    std::cout << "\n\n";

    return 0;

}
```

Output:-

```
/tmp/K5rHjIxspk.o
����������
�x.......�
�x.......�
�x...���.�
�x.�...�.�
�x.�...�.�
�.x�����.�
�..xxxx..�
�......xx�
����������

Path cost 11: (0, 0) (0, 1) (0, 2) (0, 3) (0, 4) (1, 5) (2, 6) (3, 6) (4, 6)
    (5, 6) (6, 7) (7, 7)
```