Path-Finding using Breadth First Search Algorithm

ROGÉRIO DE LEON PEREIRA[1]

[1]FourX Development Ltda (4x)
joshua@fourx.com.br

1 Introduction

A big problem for Acknex users is how to create a simple and efficient path system. In this tutorial you will learn how transform a simple set of paths (using the version A6.22 new path/waypoint design) in a graph and solve it using the Breadth First Search algorithm (BFS) with some little modifications.

2 Graph Theory

The graph theory began with the work of Swiss mathematician, *Leonhard Euler* (1707- 1783), in a problem about crossing every the *Könisberg* Bridge network across a river with an Island in between. How to traverse all bridges without crossing one bridge twice? (Figure 1)

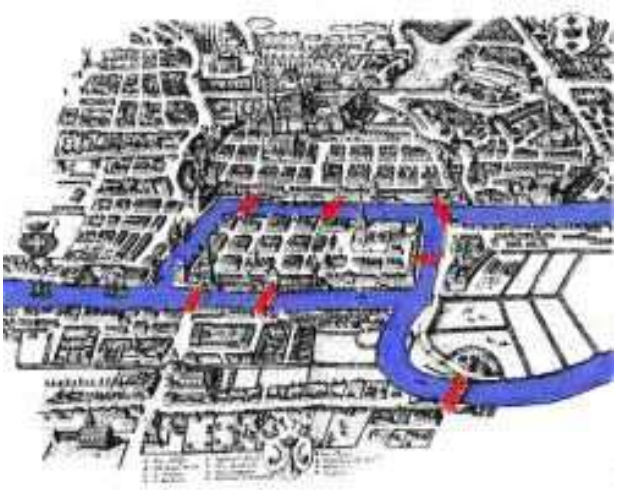


Figure 1 – *Könisberg* Bridge Network

Euler solves this problem creating a graph where earth was *node* and bridge was *edge* (Figure 2).

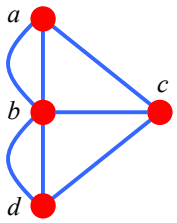


Figure 2 – Graph by Euler

Informally, a graph is a finite set of dots called nodes (or vertices) connected by links called edges (or arcs). More formally: a simple graph is a (usually finite) set of nodes V and set of unordered pairs of distinct elements of V called edges. Two nodes are adjacent if they are connected by an edge

Not all graphs are simple. Sometimes a pair of nodes is connected by multiple edges yielding a *multigraph*. At times nodes are even connected to themselves by an edge called a loop, yielding a *pseudograph*. Finally, edges can also be given a direction yielding a *digraph* (or directed graph).

A graph is connected if there is a path connecting every pair of vertices (Figure 3a). A graph that is not connected can be divided into connected components (Figure 3b).

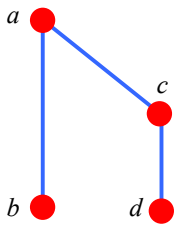


Figure 3a
Connected

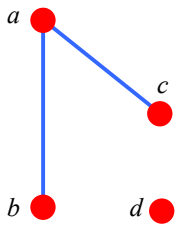


Figure 3b
Not Connected

3 Breadth First Search – BFS

One trivial technique to look for a way between two nodes in a connected graph is the *Breadth First Search* algorithm.

This technique consists in choose any starting node, search all adjacent nodes not visited and put it in an array (*FIFO – First In First Out*), and repeat this procedure while the array is not empty.

Each visited node receives as label the value from the label of the previous node increased of 1. With this, after cover all the graph it's be possible to found the way between the initially chosen node and any another node of the graph.

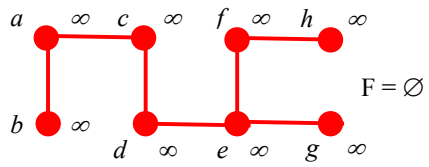
```
1   set all nodes like not marked
2   label al nodes with infinity value
3   label node v with1 and put on the FIFO
4   while FIFO array is not empty
5       make v the first node in FIFO
6       for each node w adjacent from v do
7           if w not marked
8               mark w
9               w label = v label + 1
10              put w on the end of FIFO
11      remove v of FIFO
```

Here it is an example to demonstrate graphically and step-by-step how the algorithm would work in given graph. Each node will be represented by a letter and its respective label for a number.

The nodes and the not visited edges will be painted of red and the visited nodes and edges will be painted of blue. Together to each representative figure it will also have the content of the array (FIFO) of vertices (F).

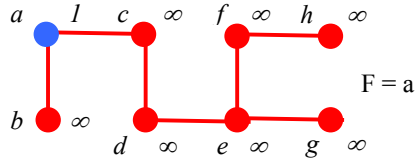
All the nodes labels will start with infinite value (∞).

Step 1



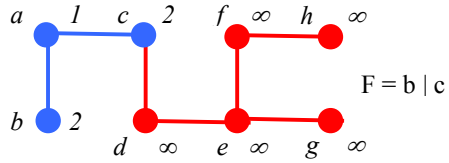
The node *a* is placed in the array and its label modified for 1.

Step 2



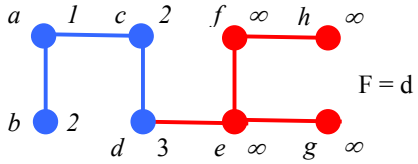
After that, the edges are covered in search of the not visited adjacent nodes.

Step 3

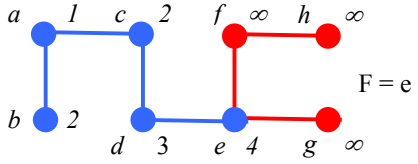


The process is repeated while the array will not be empty.

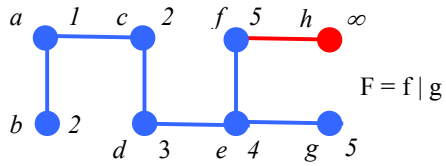
Step 4



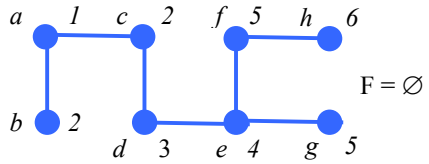
Step 5



Step 6



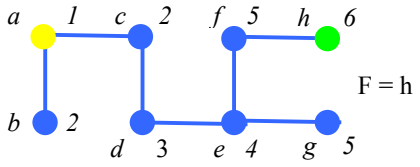
Step 7



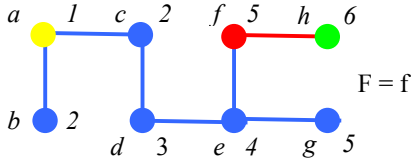
The algorithm ends when not exist more nodes in array *F*. To find the way between the initial node and any another node of the graph, just choose the destination node, select a node adjacent of lesser label, place it in an array and repeat the process until the node in the array is the origin node.

For illustration effect, the origin node will be colored of yellow, and the destination node of green. Reds will be all the nodes and edges that will be between the destination and the origin. The node *a* it was used as origin of the search, and node *h* as destination.

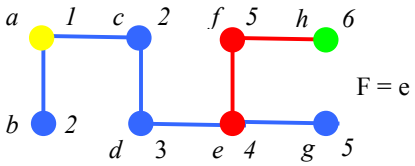
Step 1



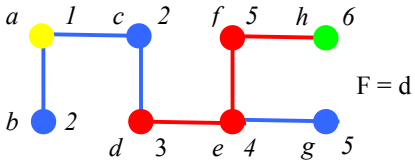
Step 2



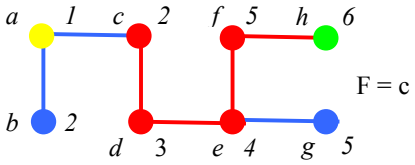
Step 3



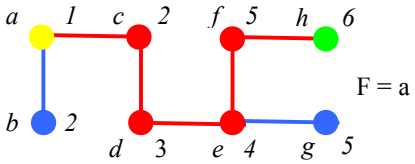
Step 4



Step 5



Step 6



The way of node *a* for node *h* is: a, c, d, e, f, h.

4 Graphs on Acknex 3D Engine

It is possible to create a graph in the A6 similar to the graph used in the example of item 3. In a new map or a map used for tests, select the menu Object / Add Path (figure 4).

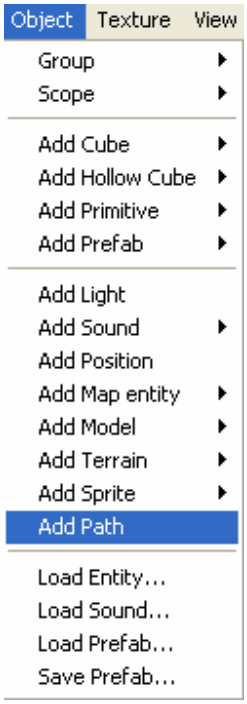


Figure 4 – Adding a Path

The WED represents the Path object as a square with a central circle (the gxl2dx8a.dll is required for path editing). After selecting a path, enter the Vertex Move Mode (Figure 5) for editing nodes and edges. The direction of every edge is indicated by a little arrow (Figure 6a).

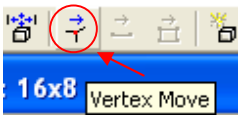


Figure 5 – Using the Vertex Move Mode

Select a node by clicking on it. The node is highlighted in red. You can drag it around with the mouse. Clicking at another place in the window creates a new node and draws an edge from the last node to the new one.

For connecting two nodes with an edge, either click on the first node, drag the *rubber band* with pressed [Ctrl] and left mouse key to the second node, and drop it there.

Or click with [Ctrl] pressed on a node to draw an edge from the previously selected node to this one. Each node can have an arbitrary number of edges to other nodes.

Touching an edge with the mouse highlights that edge (Figure 6b), right clicking on it opens an edge properties panel for setting the edge direction, bezier factor (not used yet), weight, and skill. (Figure 7).

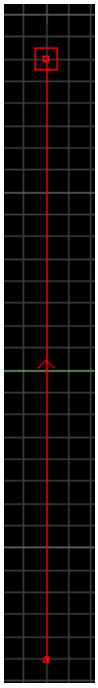


Figura 6a
Edge between 2 nodes

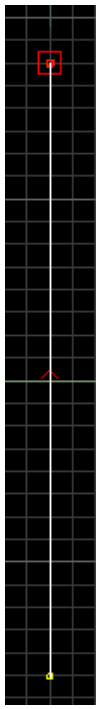


Figura 6b
Touching an edge

You can notice on the edge properties panel (Figure 7) a button called *direction* where you can define the edge direction. In this case the direction of the edge is from the node *2* to node *1*, *a = (2,1)*.

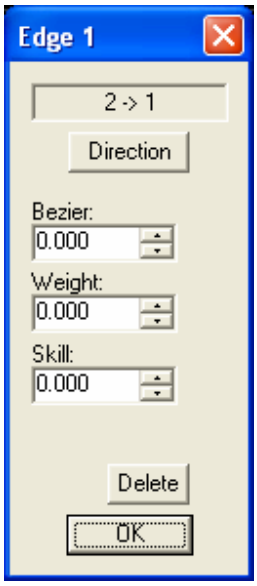


Figure 7 – edge properties panel

Click once on the *direction* button, the edge direction will change to node *1* for the node *2*, *a = (2,1)* (Figure 8).

Figure 8 – Changing the edge's direction

Click one more time on the *direction* button to change the edge for no direction (Figure 9), exactly what will be necessary for the example.

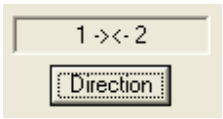


Figure 9 – Edge without direction

Now, let's add some more nodes, binding them with the edges. Don't forget to change all edges to no direction (Figure 9). The result will be close on the Figure 10.

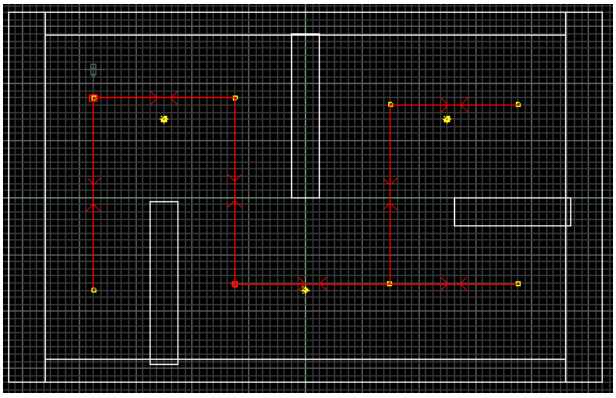


Figure 10 – Graph on the A6 map

Select any node and right clicking on it opens a node properties panel for setting the 6 node skills. They can be used in C-Script for triggering path events. (Figure 11).

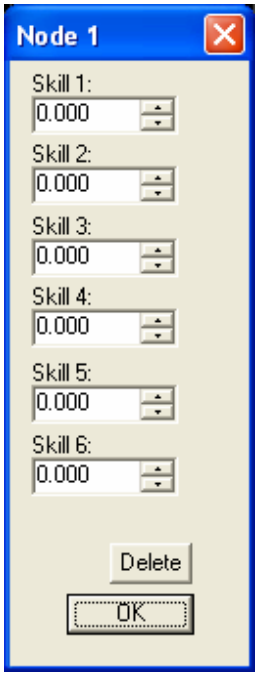


Figure 11 – Node properties panel

5 Implementation

For the given example, it is considered that each node of the graph can have a label. The nodes that will have label 1 are destined as escape way, of label 2 is for patrol way and of label 0 they are common nodes. The values of label of each node will be kept in its skill1 property.

Let's create some variables and arrays to be used in the breadth search algorithm. The general graph information will be stored on the path entity.

```
var Nodo = 1;    //current node
var Nodo_S[6];   //to sotore node skills
var Nodo_A[100]; //array for target nodes
var Nodo_AI = 0; //índice for target nodes
var Nodo_AC = 0; //counter for target nodes
var Nodo_N = 0;  //number of nodes
var Nodo_C = 0;  //counter for general use
var Nodo_T[100]; //array for node labels
var Nodo_TI = 0; //índice for node labels
var Nodo_O[100]; //path order
var Nodo_OI = 0; //índice for path order
var Fila[100];   //FIFO array
var Fila_I = 0;  //índice for FIFO
var Fila_C = 0;  //counter for FIFO
var Aresta = 0;  //number of edges
```

The breadth search algorithm is divided in two parts: the function *busca* and the function *busca_largura(v)*. All steps are described above on the item 3

```
function busca(w){
  busca_limpa();
  Fila[Fila_I] = w;
  Nodo_T[w] = 1;
  While (Fila[Fila_I] != 0) {
    busca_largura(Fila[Fila_I]);
    Fila_I += 1;
  }
  busca_ordena();
}
```

```
function busca_largura(v){
  Aresta = 1;
  while (Aresta != 0){
    Nodo = path_nextnode(my,v,Aresta);
    if (Nodo == 0) {
      Aresta = 0;
    }
    else {
      if (Nodo_T[Nodo] > Nodo_T[v]) {
        Nodo_T[Nodo] = Nodo_T[v] + 1;
        Fila_C += 1;
        Fila[Fila_C] = Nodo;
      }
      Aresta += 1;
    }
  }
}
```

Additionally, we will use more 3 functions: *busca_limpa()*, *busca_alvo(tipo)* and *busca_ordena()*. The first one is used to empty all the variables and arrays used in the process; the second function search the path entity and verifies which nodes is of a specific type, type this chosen through the parameter of the function. The last one serves to feed an array with the order of the nodes that will be covered.

```
function busca_ordena(){
  Nodo_AI = Int(random(Nodo_AC));
  Nodo_OI = Nodo_T[Nodo_A[Nodo_AI]];
  Nodo_O[Nodo_OI] = Nodo_A[Nodo_AI];
  while (Nodo_OI > 1) {
    Aresta = 1;
    Nodo = 1;
    While (Nodo != 0) {
      Nodo = path_nextnode(
             my,Nodo_O[Nodo_OI],Aresta);
      if (Nodo != 0
        && (Nodo_T[Nodo] < Nodo_OI)) {
        Nodo_OI -= 1;
        Nodo_O[Nodo_OI] = Nodo;
        Nodo = 0;
      }
      Aresta +=1;
    }
  }
  Nodo_OI = 1;
  Nodo = Nodo_O[Nodo_OI];
}
```

```
function busca_alvo(tipo){
  Nodo_C = 0;
  Nodo_AI = 0;
  while (Nodo_C < 100) {
    Nodo_A[Nodo_C] = 0;
    Nodo_C += 1;
  }
  Nodo_C = 1;
  while (Nodo_C <= Nodo_N) {
    path_nodepos(my,Nodo_C,temp);
    path_nodeskills(my,Nodo_C,Nodo_S);
    if Nodo_S[0] == tipo {
      Nodo_A[Nodo_AI] = Nodo_C;
      Nodo_AI +=1;
    }
    Nodo_C += 1;
  }
  Nodo_AC = Nodo_AI;
  Nodo_AI = 0;
  Nodo_C = 0;
}

function busca_limpa(){
  Nodo_C = 0;
  while (Nodo_C <= Nodo_N){
    Fila[Nodo_C] = 0;
    Nodo_T[Nodo_C] = Nodo_N;
    Nodo_O[Nodo_C] = 0;
    Nodo_C += 1;
  }
  Fila_I = 0;
  Fila_C = 0;
  Nodo_C = 0;
  Nodo_TI = 0;
  Nodo_OI = 0;
}
```

6 Final Touch

When playing a game, the player wants a good entertainment and a lot of fun. To provide challenge is something very important to be planned in a computer game. The breadth first search algorithm is a simple and very fast technique to look for a way between two nodes in a given graph.

As seen in the example of this tutorial, a graph can be used to create diverse routes of escape, places of patrol etc. With this it can be made with that the controlled entities for artificial intelligence always act with bigger realism choosing possible ways taking rational decisions in accordance with the interaction of the player.