

A* Pathfinder in Visual Basic

with the A-star algorithm

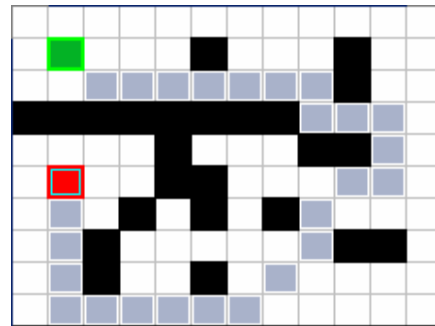
Description

This guide explains the use of the subroutine **Pathfinder_A_star** performing the 2D pathfinder. Conceptually speaking, a pathfinder is quite simple: it is an algorithm of finding a route on the ground from a starting location to a target location, avoiding the obstacles and minimizing the cost. But its implementation can be quite complicated.

This Visual Basic routine implements the so called A-star algorithm, also abbreviated as A*.

It works on a rectangular ground plane, called the search region or map, divided in little squares, called cells or nodes.

The cells can be walkable, with different level of difficult, or unwalkable (for example: walls and holes). Each cell has a cost, associated to its level that contributes to the global cost of the entire path. This algorithm searches for a path having minimum cost.



Specification

Pathfinder with A* algorithm
v.1.0 3-4-2005, by Foxes Team

```
Sub Pathfinder_A_star(Map(), PathStart(), PathEnd(), Path(), ErrMsg, Optional Score,
Optional Stat)
```

Input

```
Map = array(N,M), contains the mapping of the ground
    0 = walkable, 1 walkable with difficult 1, 2 with difficult 2, etc.
    -1 = unwalkable
PathStart = array(2), indicates the starting point
PathEnd = array(2), indicates the ending point
```

Output

```
Path = array(*,2), contains the nodes of the path nodes, from end to start
ErrMsg = error message
Score = array(*,5) containing the score F, G, H of each explored node
Stat = array(6) containing same statistics:
    Stat(1) = Nodes Explored
    Stat(2) = Nodes of the path
    Stat(3) = Efficiency
    Stat(4) = Estimated cost
    Stat(5) = Effective cost
    Stat(6) = Difficulty
```

Map: The input parameter Map contains the ground information. It is an (N x M) array. Each element represents a single square of the ground. Each square has a level from 0 to 99 indicating the crossing difficult. The negative level -1 indicates an unwalkable ground (holes or walls)

The user provides the starting cell (I_s, J_s) and the target cell (I_t, J_t) by two arrays: PathStart and PathEnd

The routine searches for the best path and returns the cells of the path, in reverse order, in the Path array; it has two columns [i, j] and several rows, depending to the path-length. The first element is the target cell, the last one is the starting cell.

If the algorithm does not converge, an error is returned in the variable ErrMsg

In addition, the routine returns the list of the cells explored with their score F, G, H. Score is an array of 5 columns [i, j, F, G, H] and several rows depending on the cells explored. (See the *algorithm details* section)

The routine returns also several useful statistics about the elaboration performed:

Cells explored:	the cells of the map effectively explored
Cells of the path:	the cells used by the path or, in other worlds, the path-length
Efficiency:	an index defined as (path-length)/(cells explored)
Estimated cost:	the cost of the path estimated at the beginning
Effective cost:	the cost measured at the end of the path
Difficulty:	an index defined as (Effective cost)/(Estimated cost)

Usage

Driver programs for this routine depend strictly by the application that you want to implement and by the environment that you are using. At the *examples* section you can find a complete macro developed in Excel.

Variable declaration

Anyway the declaration variables section is always the same. You have to declare the arrays:

```
Dim myMap(), PathStart(), PathEnd(), Path(), ErrMsg, Score, Stat
```

You have to provide the dimensions only for the input arrays:

```
ReDim PathStart(1 To 2), PathEnd(1 To 2)
ReDim myMap(1 To N, 1 To M)
```

The other arrays are managed by the routine pathfinder

Algorithm Details

A good explanation, with several tips and tricks, can be found at the references section links. Here we only want to explain the scoring method, that it is the heart of the A* algorithm. It assigns a global score or cost, called F, at each visited cell of the map. The global cost is the sum of two other costs, called G and H:

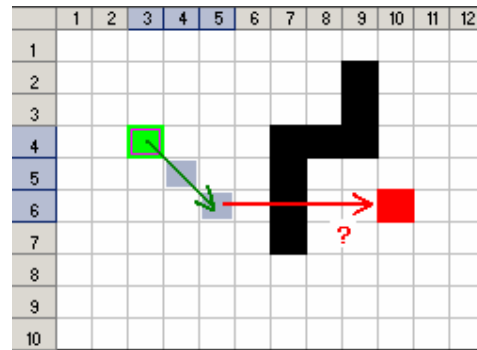
$$F = G + H$$

where

G = the movement cost to move from the starting point A to a given cell B on the grid, following the path generated to get there.

H = the estimated movement cost to move from that given cell on the grid to the final destination, point B, ignoring any obstacle on the path. Of course, we really don't know the actual distance until we complete the entire path. So we can only estimate this score using several approximated method. You are given one way to estimate H in this document, but there are many others that you can find in other articles on the web.

In this example the starting cell (green) is (4,3) and the target cell (red) is (6,10). Black cells are unwalkable (a wall for examples). The current position of the walker is (6,5) that it is the last cell of the path (light-blue cells). The path done is represented by the green arrow; the path to do is represented by the red arrow. Note that the walker cannot know exactly this path. He can only estimate it by his current position and the final position to reach. The cost of path can be deduced assuming the following table

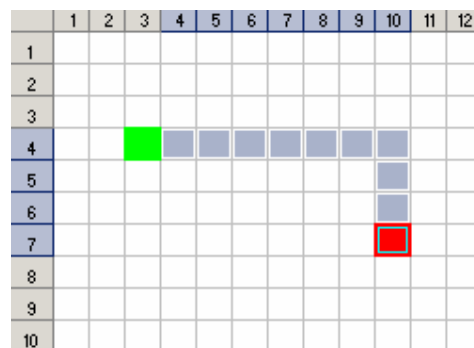


horizontal and vertical movement	10 for each cell
diagonal movement	14 for each cell

So the costs can be calculated as $G = 14 * 2 = 28$, $H = 10 * 5 = 50$, $F = 28 + 50 = 78$. The algorithm attributes the score $F = 78$ to the cell (6, 5) and repeat this computation for each adjacent cell. It chooses the cell with the lowest score and move to it, adding it to the path.

One easy way for estimated H is assuming a rectangular path, also called Manhattan distance, moving only horizontally and vertically and ignoring the obstacles that are in the way.

$$H = 10 \cdot (|I_2 - I_1| + |J_2 - J_1|) = 10 \cdot (|7 - 4| + |10 - 3|) = 100$$



Variable region cost. The exploring region can have cells that are walkable, but at a higher movement cost. For example: hills, sands, rivers, taxes, tolls, etc. This problem is easily handled by adding a level to the map.

For the routine Pathfinder_A_star, the flat ground has level = 0 (the minimum). Higher levels indicate a sort of difficulty that increases the total cost of the path.

The extra cost is calculated as:

$$G = G + 10 \text{ Level}$$

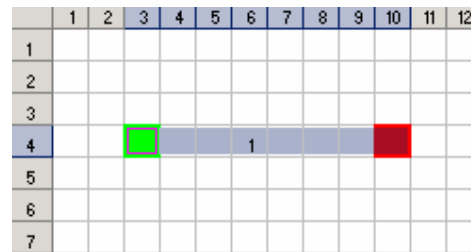
for each cell of the path

In this example the cost of the path is

$$G = 10 \cdot (10-3) + 10 \cdot 1 = 70 + 10 = 80$$

When the region has a variable cost, the Manhattan distance method can easily fail.

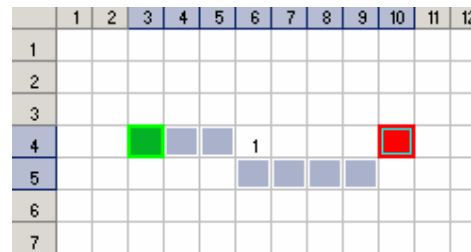
In this example the A* algorithm assumes the straight line as the best path, but it is not exact.



As shows in the figure, there is another path less expensive.

$$G = 10 \cdot 5 + 2 \cdot 14 = 78$$

In this case, following a path that avoids the cell (4,6), we have a better cost.



For this reason, the routine Pathfinder_A_star adopts an improved method for estimating the H score

Improved H score estimation formula

$$H = (14-a) \cdot |I_2 - I_1| + a \cdot |J_2 - J_1|$$

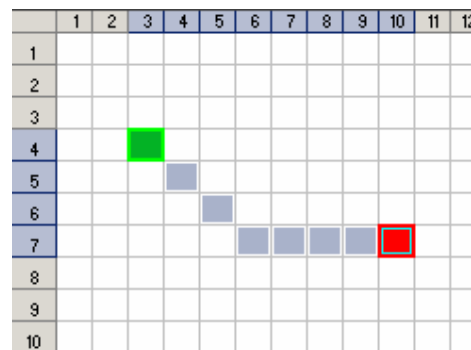
where

$$a = 10 \text{ if } |I_2 - I_1| < |J_2 - J_1|, \\ \text{otherwise } a = 4$$

In this example the estimated path is

$$H = (14-10) \cdot 3 + 10 \cdot 7 = 82$$

That it is more precise than the one obtained with the Manhattan distance (100)



This method improves the accuracy of the A* algorithm without degrading its performance.

References

"A* *Pathfinding for Beginners*", by Patrick Lester, April 21, 2004
<http://www.policyalmanac.org/games/aStarTutorial.htm>

"A* *Pathfinder Algorithm*". by Warren Willmey, 25/4/2003
<http://realmdesigns.ruski.net/?command=tutor&t=3>

Examples of the Pathfinder algorithm

To study the characteristic of the A* pathfinder algorithm comes in handy the Pathfinder1.xls, an Excel file containing a macro that uses the routine Pathfinder_A_star

A didactical pathfinder in Excel

The graphical interface is quite simple and immediate

The region consists of a rectangle of (10 x 12) cells, situated in the upper left corner. Just below there are the score matrices F, G and H and at the right the statistics table

The screenshot shows the Pathfinder Excel interface. At the top, a grid of 10x12 cells is displayed. A green cell at (3,2) is the starting point, and a red cell at (5,10) is the target point. A path of light-blue cells connects them, avoiding black obstacles. To the right of the grid is a statistics table:

cell explored	51
cell used	12
efficiency	24%
estimated cost	88
effective cost	114
difficulty	1.3

Below the grid are three score matrices:

F score	G score	H score
130 116 110 104 104 104 110 116 122 128 142	24 20 24 28 38 48 58 68 78 88 98	106 96 86 76 66 56 52 48 44 40 44
116 102 96 96 96 96 96 102 108 114 128	14 10 14 24 34 44 54 64 74 84 94	102 92 82 72 62 52 42 38 34 30 34
108 88 88 88 88	10 0 10 20 30	98 88 78 68 58
108 94 88 88 88	14 10 14 24 34	94 84 74 64 54
114 100 94 88 88	24 20 24 28 38 48	90 80 70 60 50 40
128 114 108 102	34 30 34 38 58	94 84 74 64 44
		48

The cell colored in green is the starting point; the red is the target point. The path is highlighted in light-blue. Unwalkable cells must be colored in black

In the example, the statistics summary shows that the path is composed by 12 cells and has a cost of 114. You can verify that this is truly the minimum cost.

For building this path, the A* algorithm has explored 51 adjacent cells obtaining an efficiency of 24%. Usually the efficiency varies from 10% to 30%. At the beginning, the estimated cost of the path was 88, while at the end the total path-cost is 114.

cell explored	51
cell used	12
efficiency	24%
estimated cost	88
effective cost	114
difficulty	1.3

Of course, the extra cost is due to the obstacles encountered on the path. The ratio between the total cost and the estimated cost is a raw index of the difficulties encountered. In this example is about 1.3. "Efficiency" is related to the algorithm and measures its ability to find the right path using as less explorations as possible. More efficiency means more fast.

"Difficulty" is intrinsically related to the map and measures the density of obstacles along the path. A labyrinth, for example, has a high level of difficulty, while the free plain ground has a low level of difficulty

We'll see that, fortunately, A* algorithm becomes more efficient just when the map is more difficult

Now come back to the example and try to move the target from the cell (5,10) to the cell (6,10)

Select the red cell and drag it just one cell down.

Now the total cost of the path is incremented of 10, giving

$$G = (\text{previous cost}) + 10 = 114 + 10 = 124$$

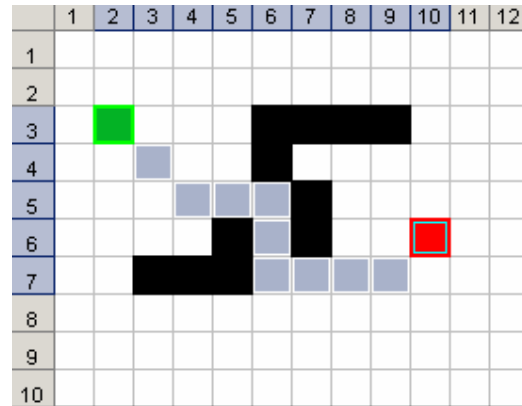
Now restart the algorithm.

Surprisingly the A* algorithm has discovered another path, across the wall, having less cost

$$G = 112$$

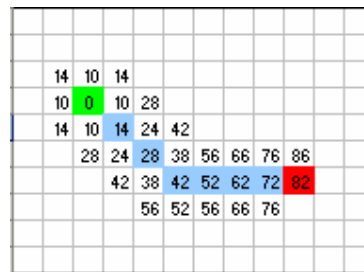
You can verify that this is truly the path with the minimum cost.

The efficiency is decremented a bit. This shows that the algorithm has worked a lot for finding this new path (and it is reasonable)



The A* algorithm is an efficient and robust algorithm for general purpose. To discover how it takes this efficiency let's arrange this example: move the green cell to (4,3) and the red cell to (7,10). Remove any dark cell from the area and start the macro. Of course this is an easy problem to solve but it is interesting to observe the cells explored by the algorithm together with the path and the G score.

cell explored	32
cell used	8
efficiency	25%
estimated cost	82
effective cost	82
difficulty	1.0

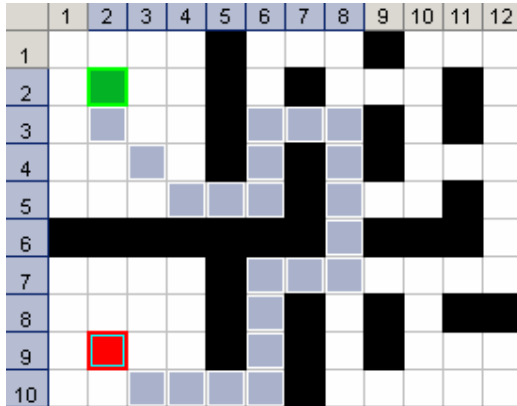


As we can see the algorithm tends to restrict the exploration area, without wasting time in explorations far from the "good" path. Only when necessary it spread up the exploration region in order to reach the goal of the best path. This feature, together with its intrinsic robustness, assures a general good performance in almost every situation.

Sometime the searched path doesn't exist or the algorithm cannot find it. In that case the macro shows an error message

The labyrinth

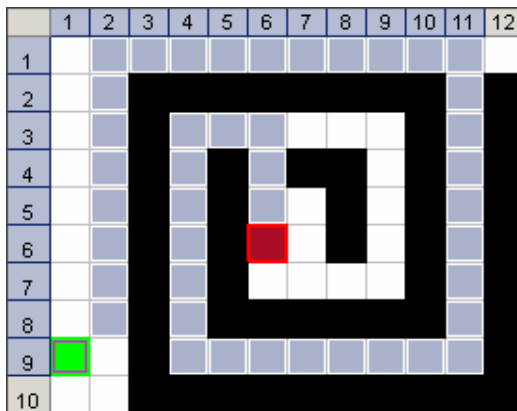
As told, the A* algorithm becomes more efficient when the situation is more difficult. This is another reason explaining because it is so largely used. For example let's see how it works with the following map



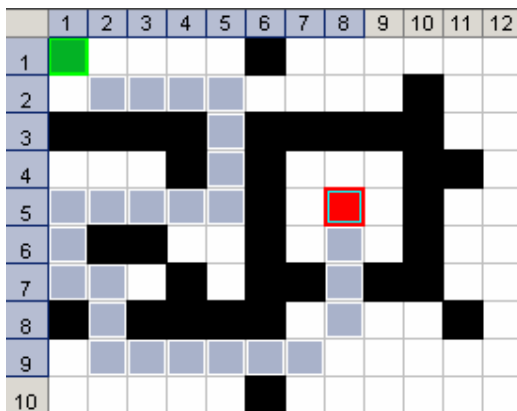
cell explored	55
cell used	23
efficiency	42%
estimated cost	70
effective cost	232
difficulty	3.3

As we can see the efficiency is increde to 42% even if the map difficulty is about 3.3

Let's see other more intricate situations



cell explored	65
cell used	44
efficiency	68%
estimated cost	62
effective cost	434
difficulty	7.0



cell explored	79
cell used	26
efficiency	33%
estimated cost	86
effective cost	258
difficulty	3.0

Try to remove the obstacle (8,11)

Variable ground cost

Many times the cost of the path is not constant along the ground, but changes for several different reasons: hills, sands, rivers, taxes, tolls, etc. The crossing-cost of each cell is given by the map-level "L" using the following formula:

$$\text{Crossing-Cost} = 10 * L.$$

In the previous example we have used a flat ground with $L = 0$.

In the following example we add a specific level to the map for simulating a hill. We see that the best path is not a straight line even if there are no obstacles on the ground. The path rounds around the hill, running on the flat ground. The total cost is 114; every other path is more expensive.

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												

cell explored	64
cell used	10
efficiency	16%
estimated cost	90
effective cost	114
difficulty	1.3

Now insert 1 in the cell (2,6) and restart the macro

	1	2	3	4	5	6	7	8	9	10	11	12
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												

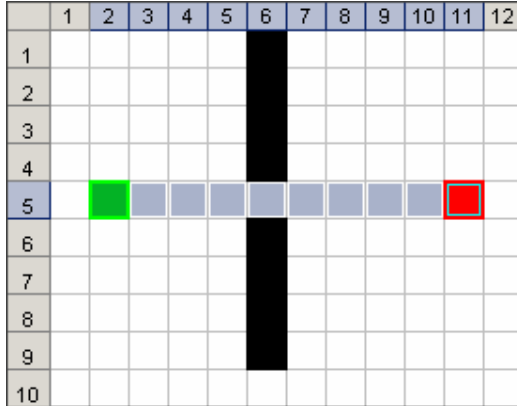
cell explored	64
cell used	10
efficiency	16%
estimated cost	90
effective cost	116
difficulty	1.3

Now the best path climbs a little on the hill instead of rounding around it. In this case the cost is 116; along the flat ground the cost is 122 or more.

As we can see, the algorithm estimates very well when it is convenient to cross a terrain or follow an alternative way.

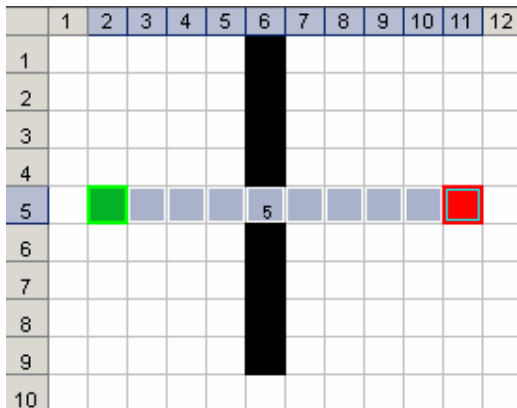
The Ticket

The map level can simulate any additional cost of the path. In the specific example we used it for take in account a ticket. Let's see. Two cities, green and red, are separated by a river having two bridges at (5,6) and (10,6). Of course, people cross the first bridge because is the most suitable



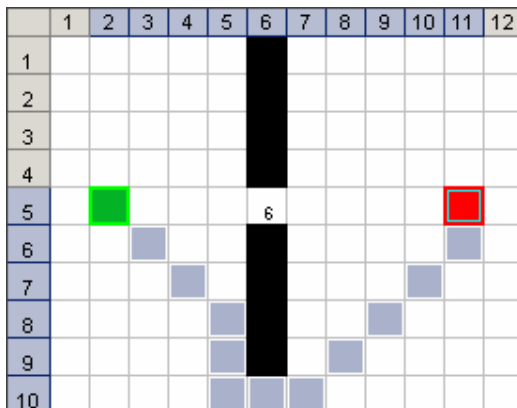
cell explored	30
cell used	10
efficiency	33%
estimated cost	90
effective cost	90
difficulty	1.0

Now, someone thinks to put a ticket for the first bridge. Begin to insert a number, from 1 to 5 in the cell (5,6) and start the macro. People still use the first bridge, with a comprehensible bad mod because the cost has now reached 140. Note also that the efficiency is quite lower. This means that the algorithm tries to discover alternative paths, before paying the exorbitant ticket!



cell explored	64
cell used	10
efficiency	16%
estimated cost	90
effective cost	140
difficulty	1.6

But, if you try with a ticket > 5, the best path changes suddenly. With the last price rise the path-cost would be 150, but people prefer to cross the second bridge because the path-cost is 148

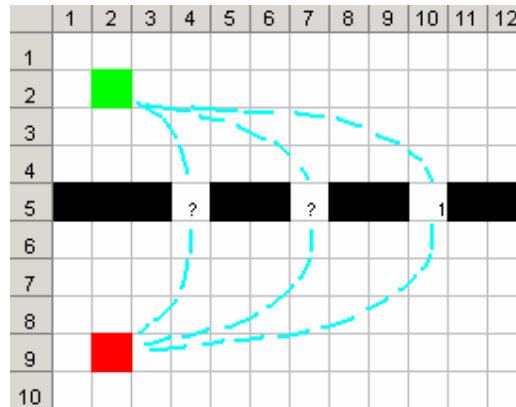


cell explored	73
cell used	13
efficiency	18%
estimated cost	90
effective cost	148
difficulty	1.6

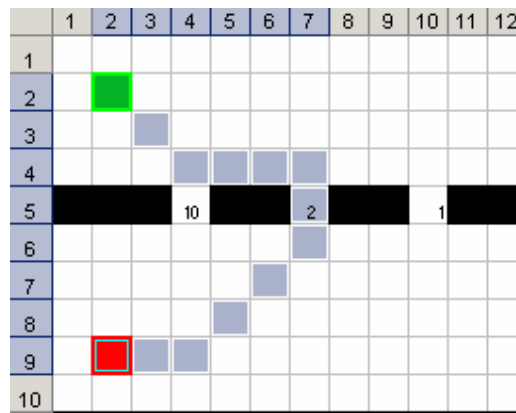
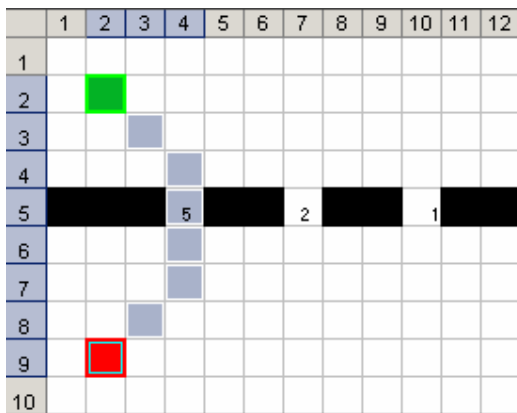
The three gates

This example is an extension of the previous one and solves the problem of the right cost of a good in a geographic scenario

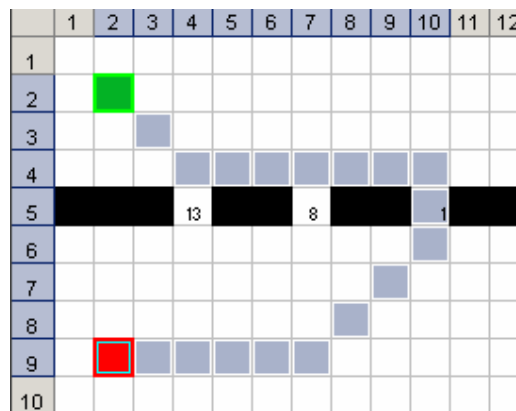
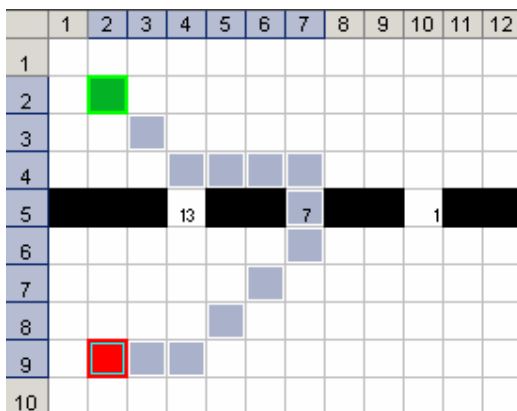
Assume to have two towns divided by a mountains chain with three gates where the people can buy food for the journey. Assumed as 1 the food cost at the 3rd gate, we want to estimate the cost at the other gates
 The food costs are related to the correspondent paths, because a short path can tolerate a higher cost and vice versa. The simulation begins inserting two hypothetic costs into the cells (5,4) and (5,7). For example 5 and 2.
 Then run the macro and observe where the path crosses.



For maximizing the profit, increase, unit-by-unit, the cost where the path crosses



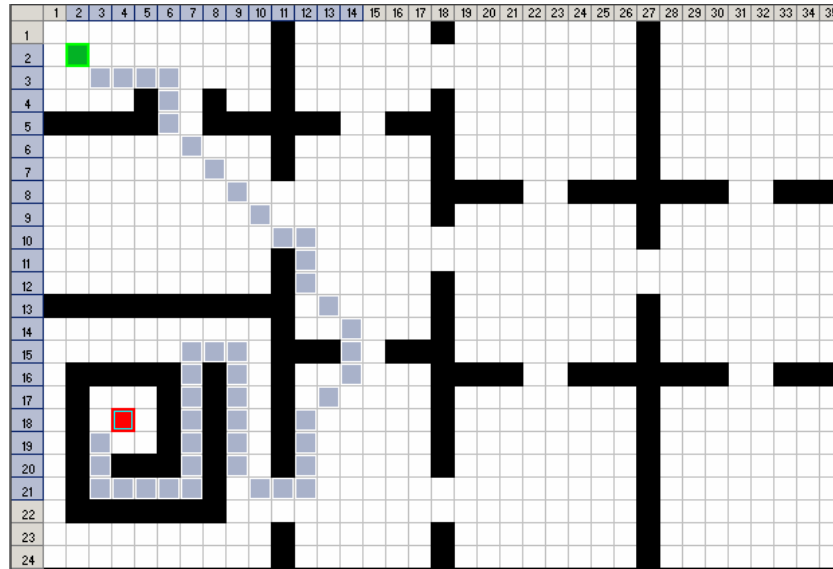
Repeat this process until the path will cross the 3rd gate



From this simple simulation we get that the maximum costs for the 1st and 2nd gates are 13 and 8 respectively, for the given geographic situation

A larger area

The macro works over a fix rectangular region of (10 x 12) cells. If you like, you can play to build a larger area that you need. The file Pathfinder2.xls for example works on a ragon of (24 x 35). Try it and have fun with.



Leonardo Volpi
Foxes Team
Italy
April, 2005