Figure 1a [Back to Article]

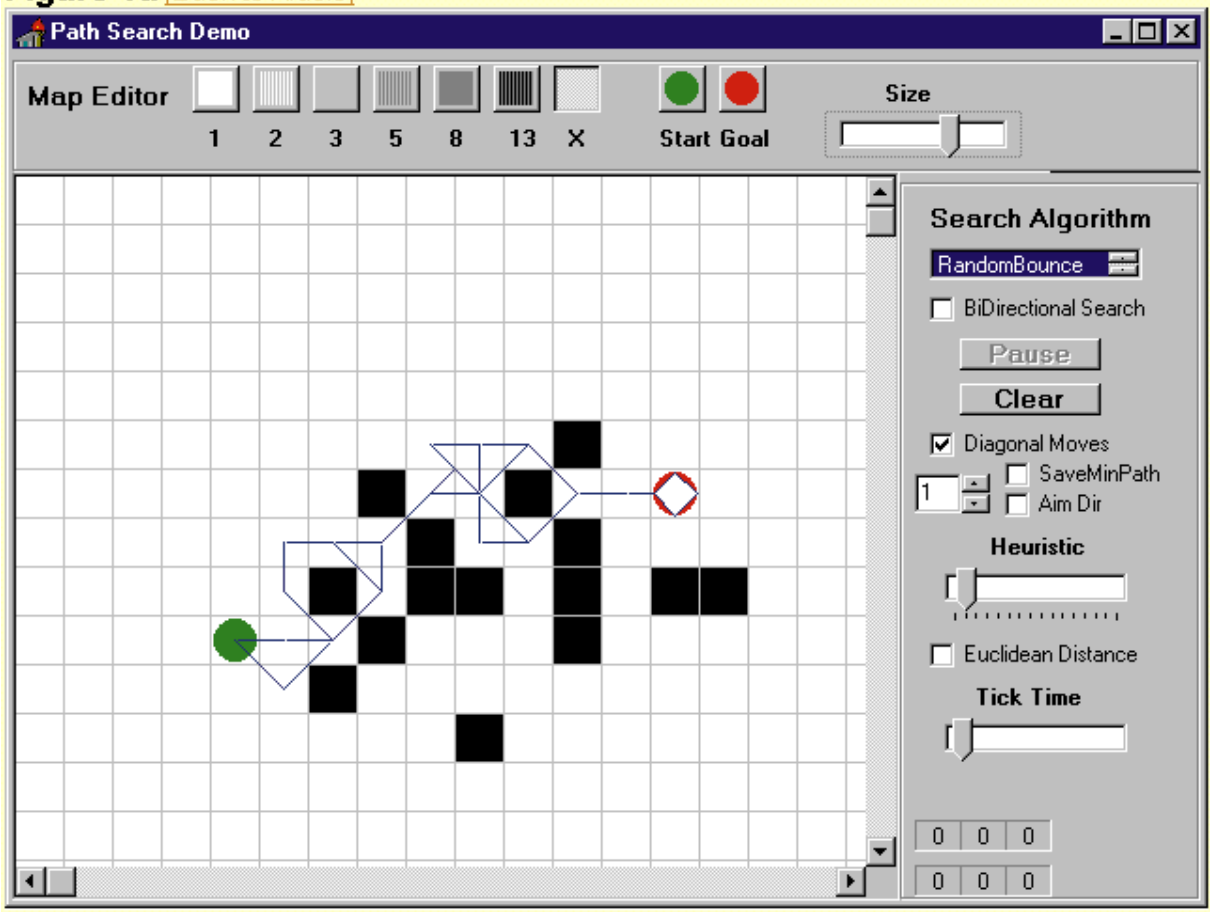


Figure 1b [Back to Article]

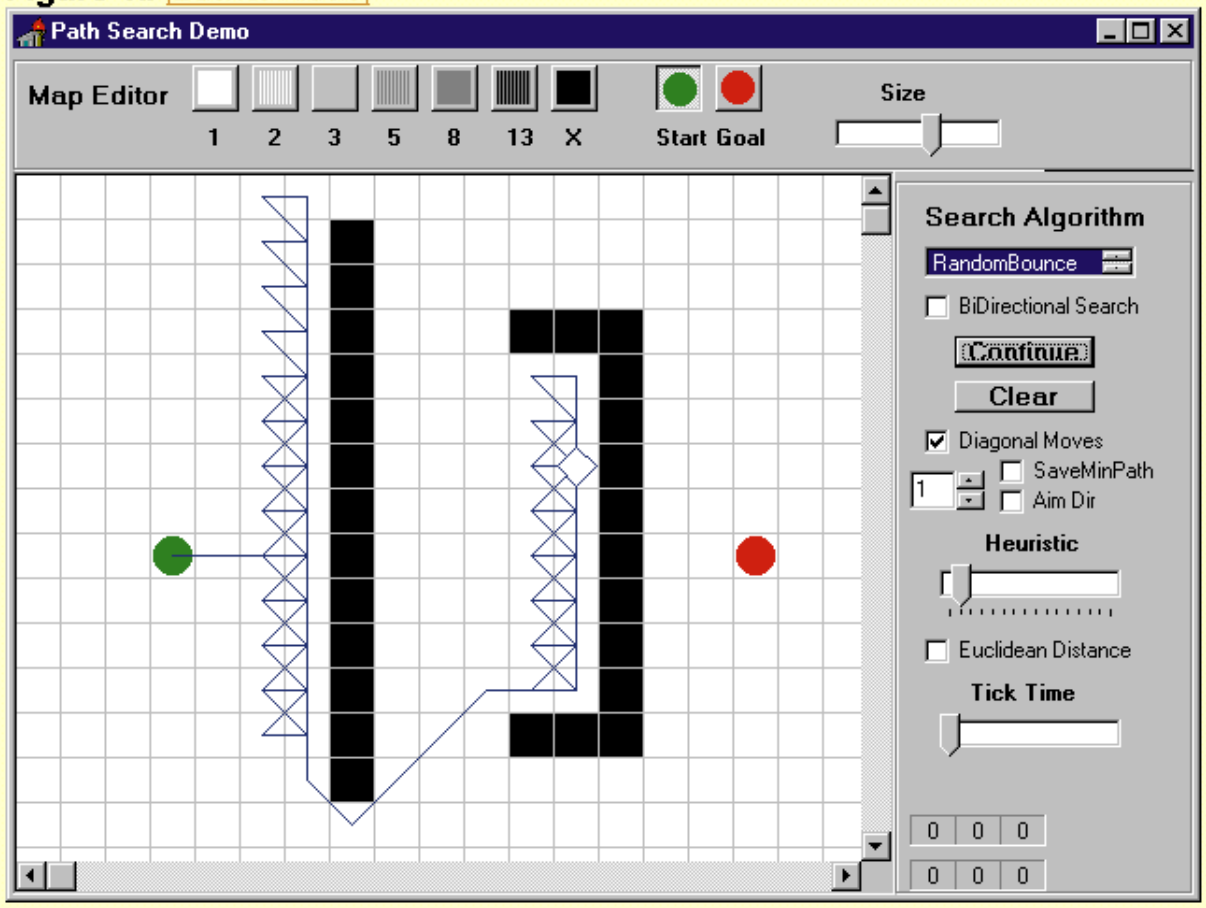


Figure 2a [Back to Article]

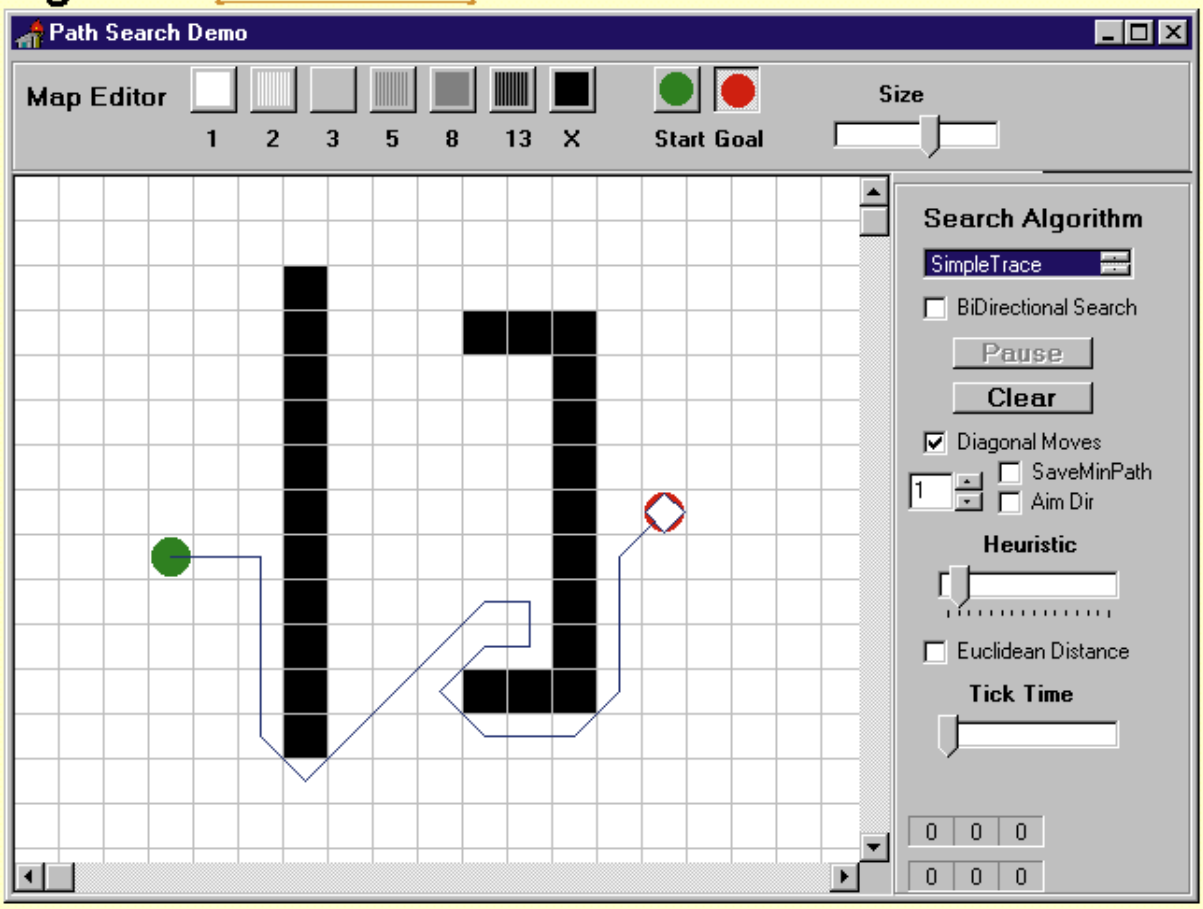


Figure 2b [Back to Article]

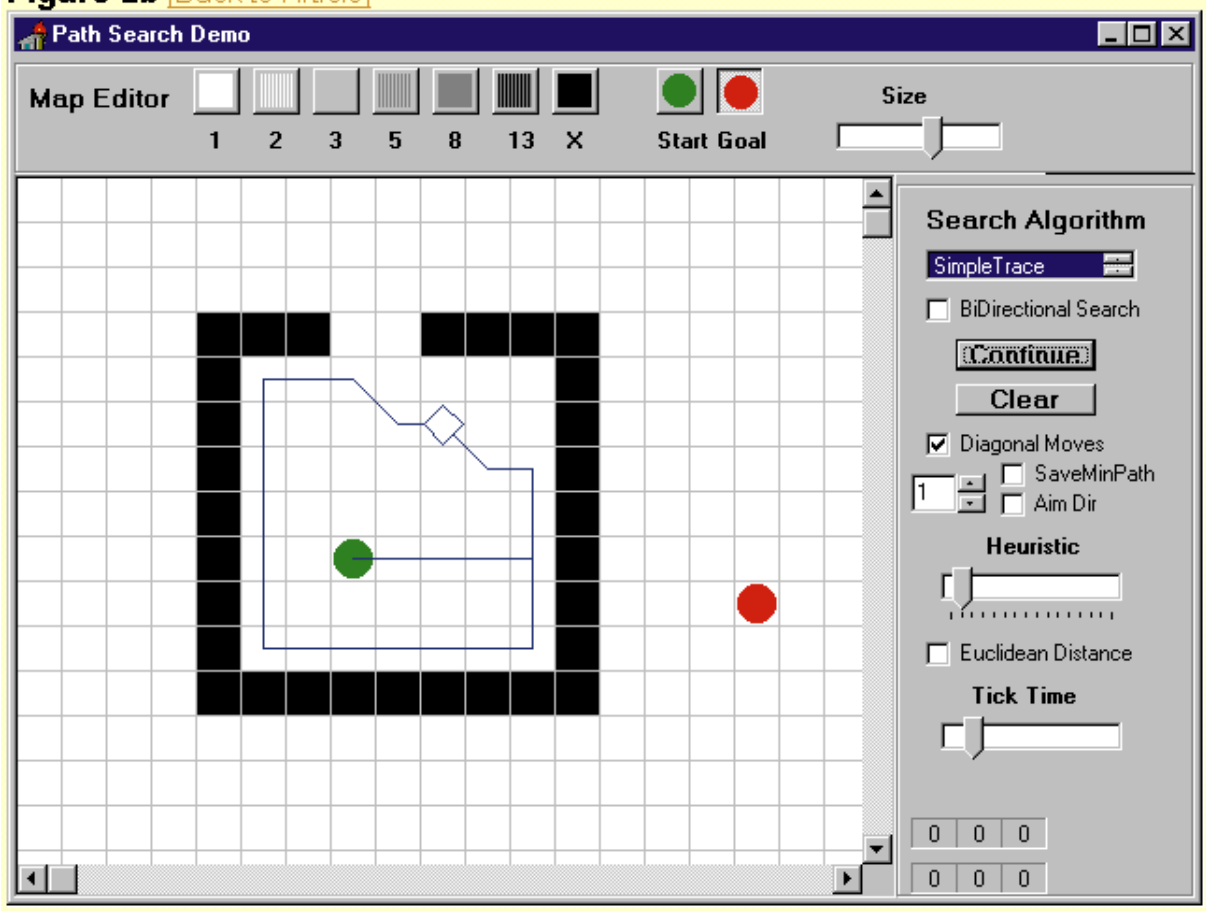


Figure 3a [Back to Article]

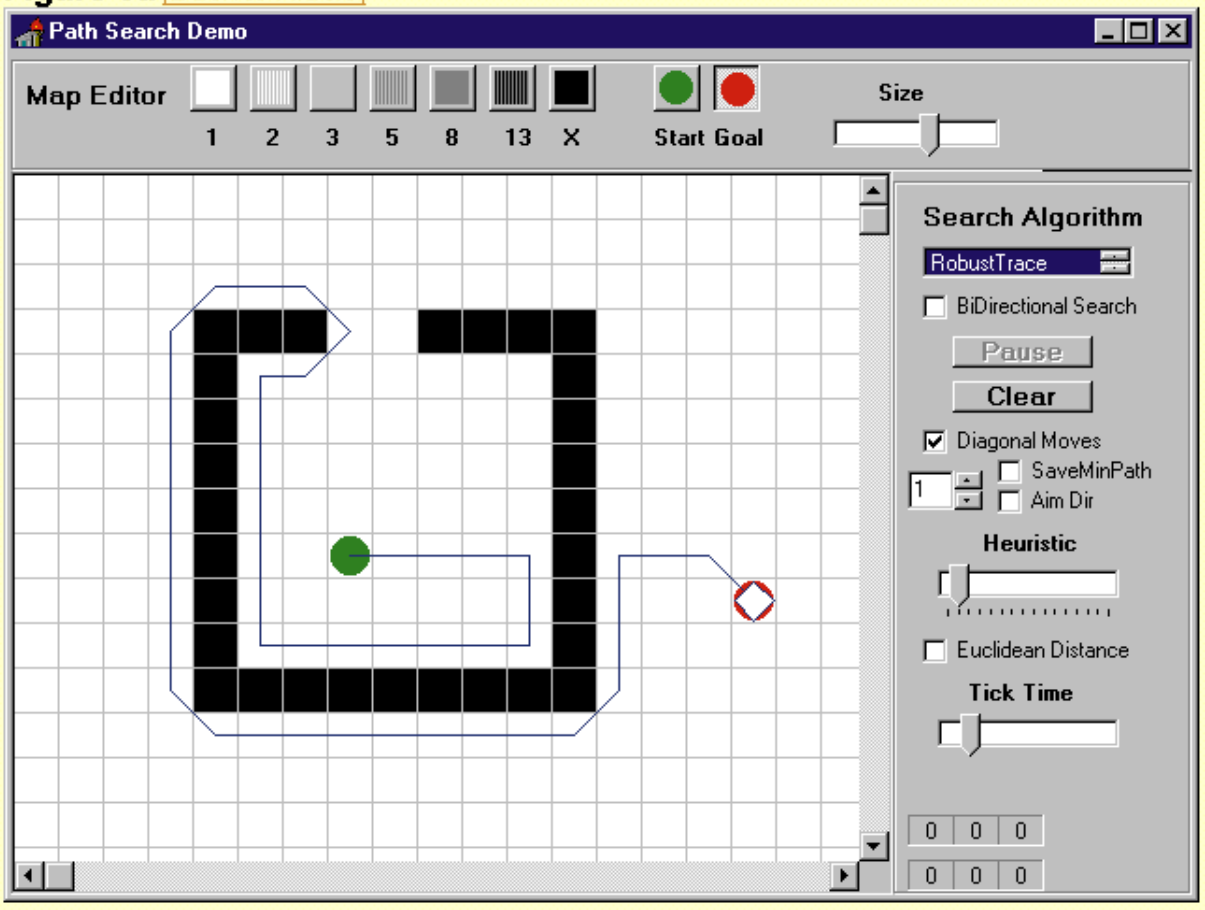


Figure 3b [Back to Article]

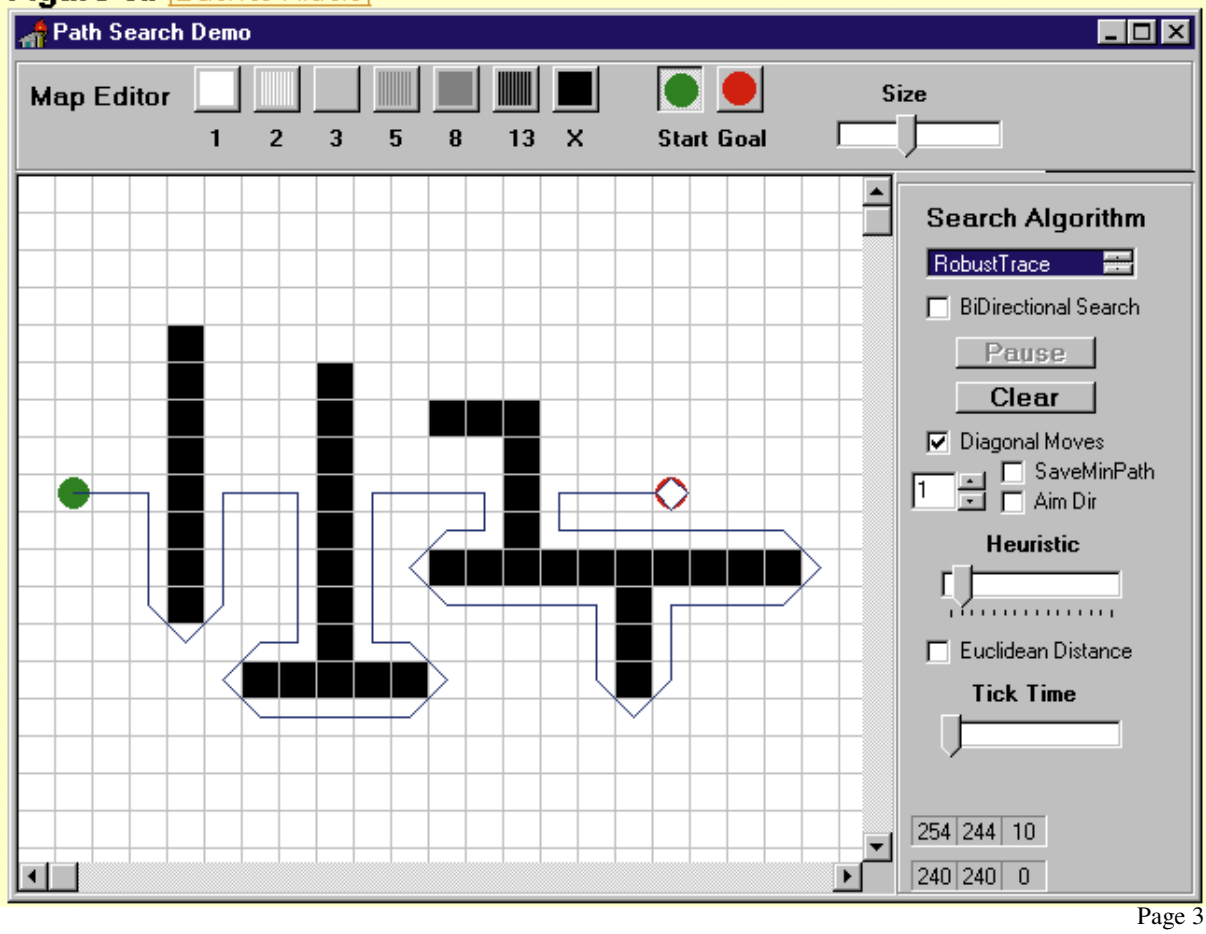


Figure 4 [Back to Article]

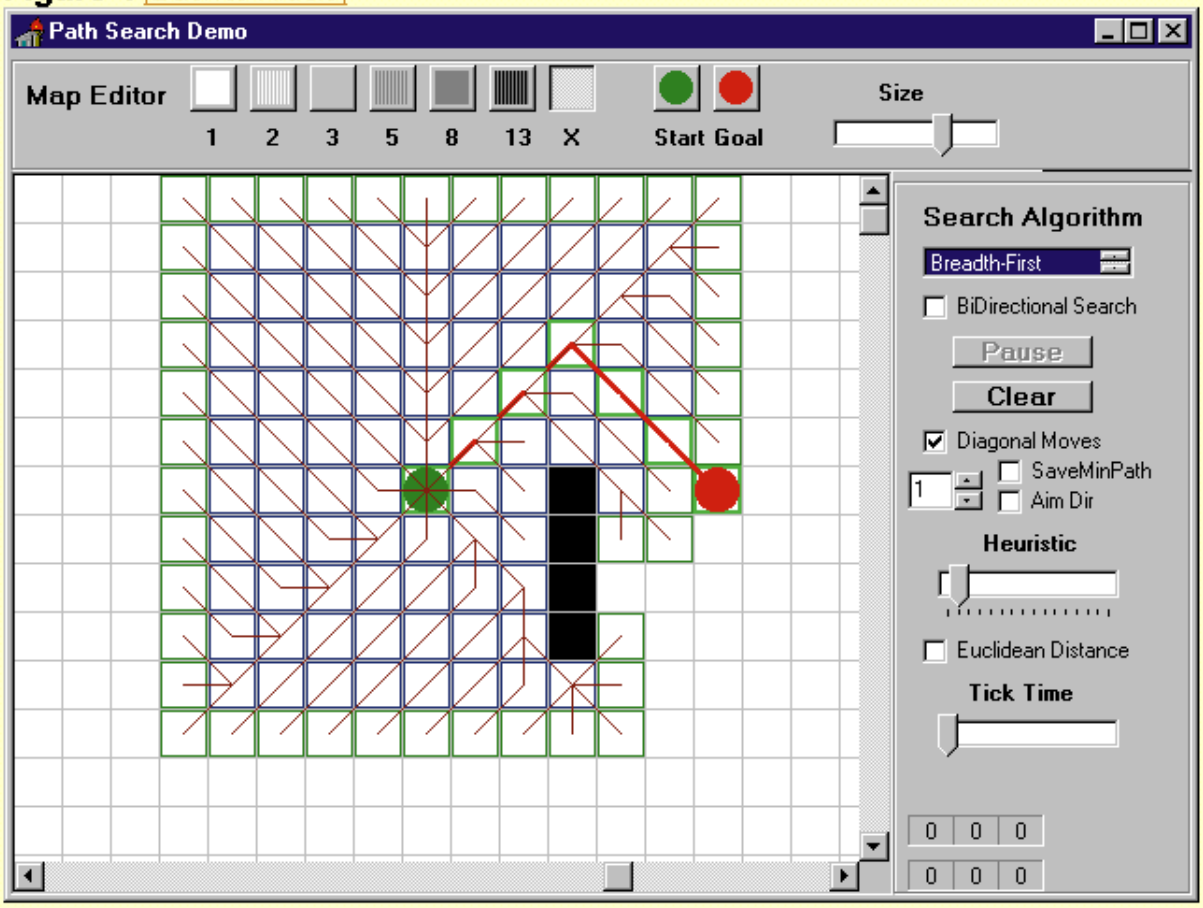


Figure 5 [Back to Article]

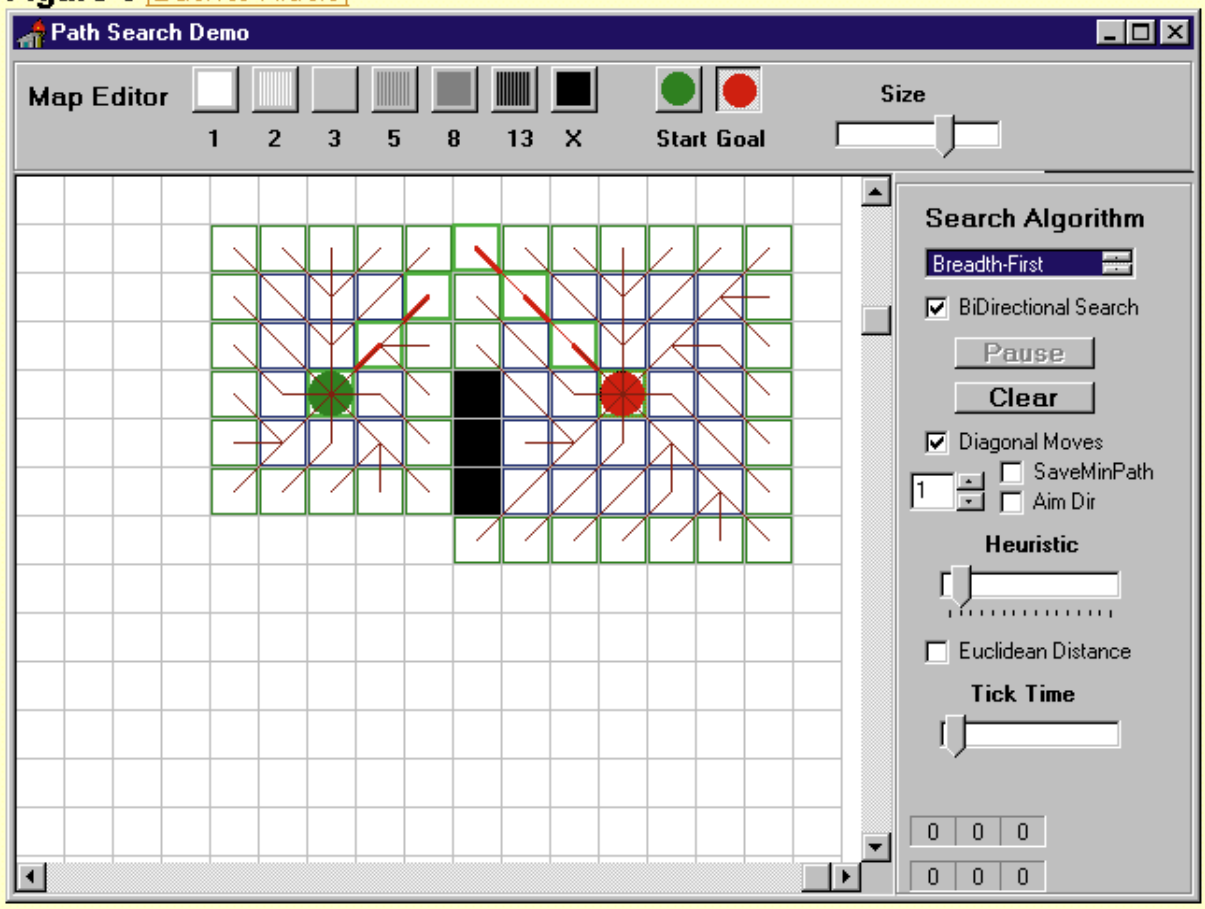


Figure 6 [Back to Article]

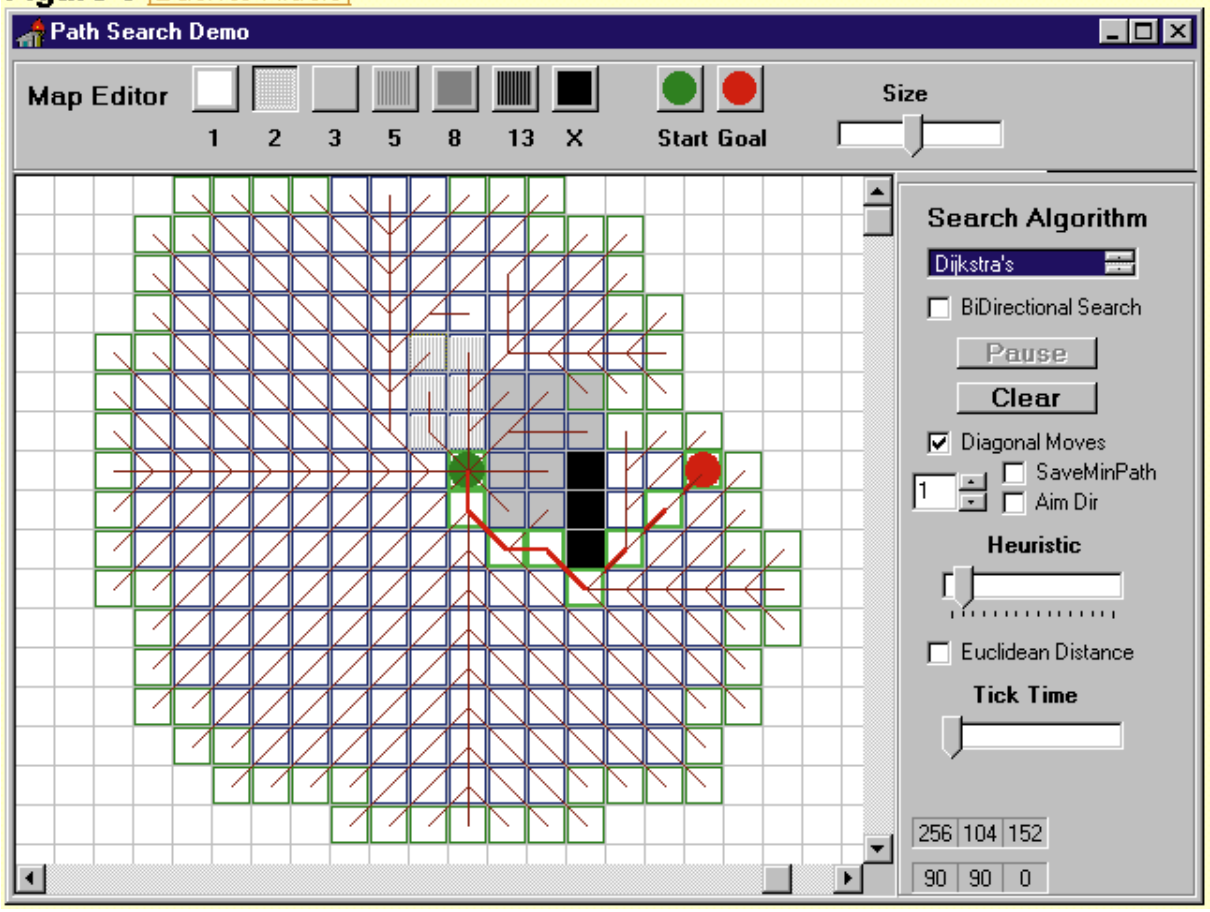


Figure 7 [Back to Article]

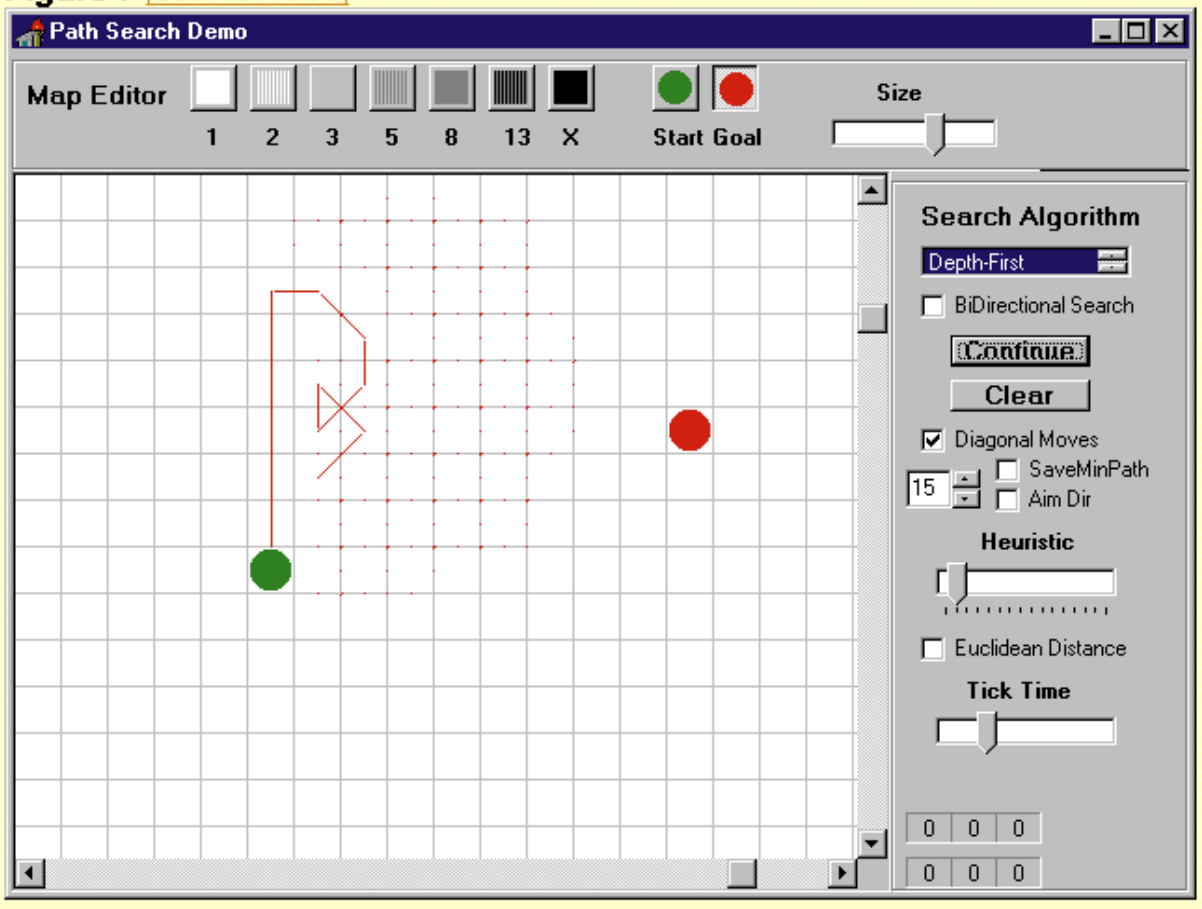


Figure 8a [Back to Article]

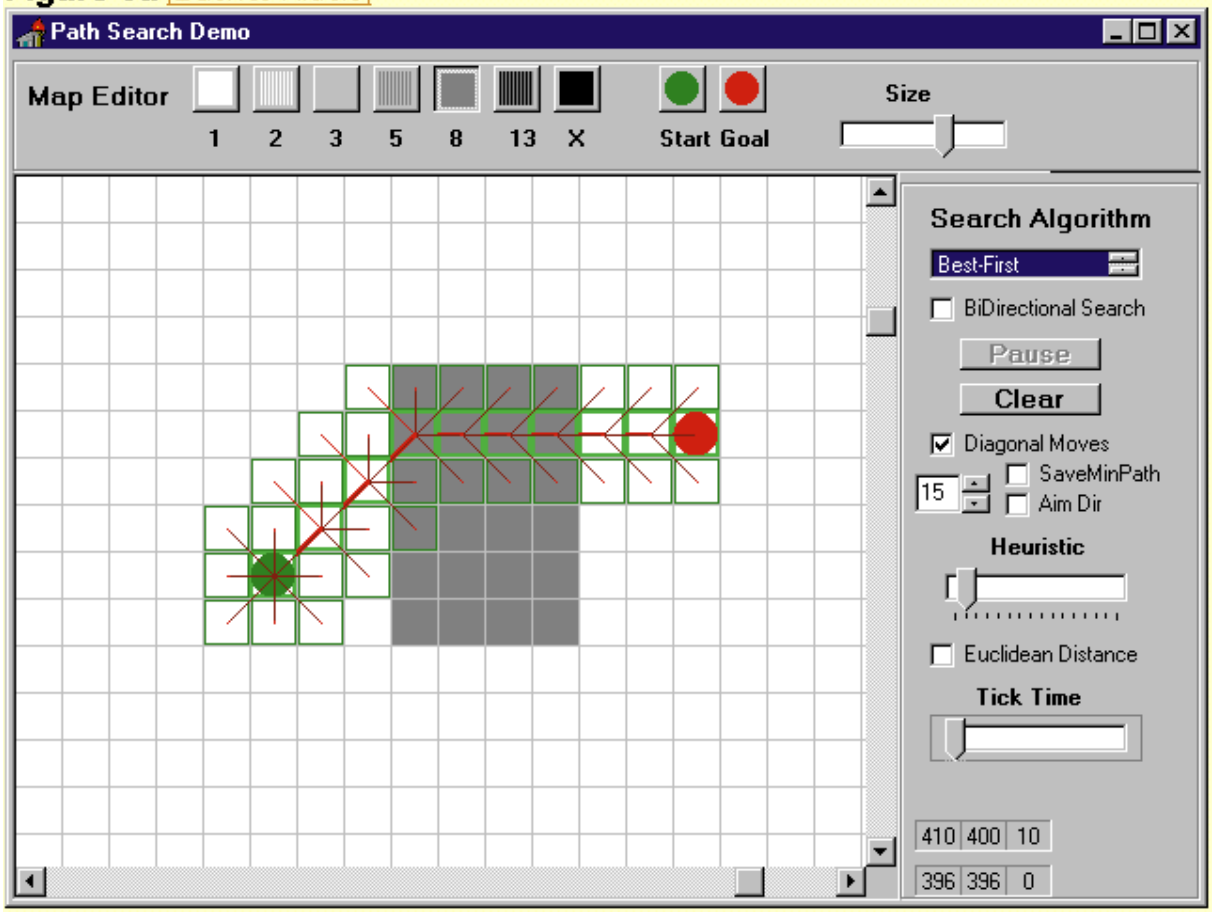


Figure 8b [Back to Article]

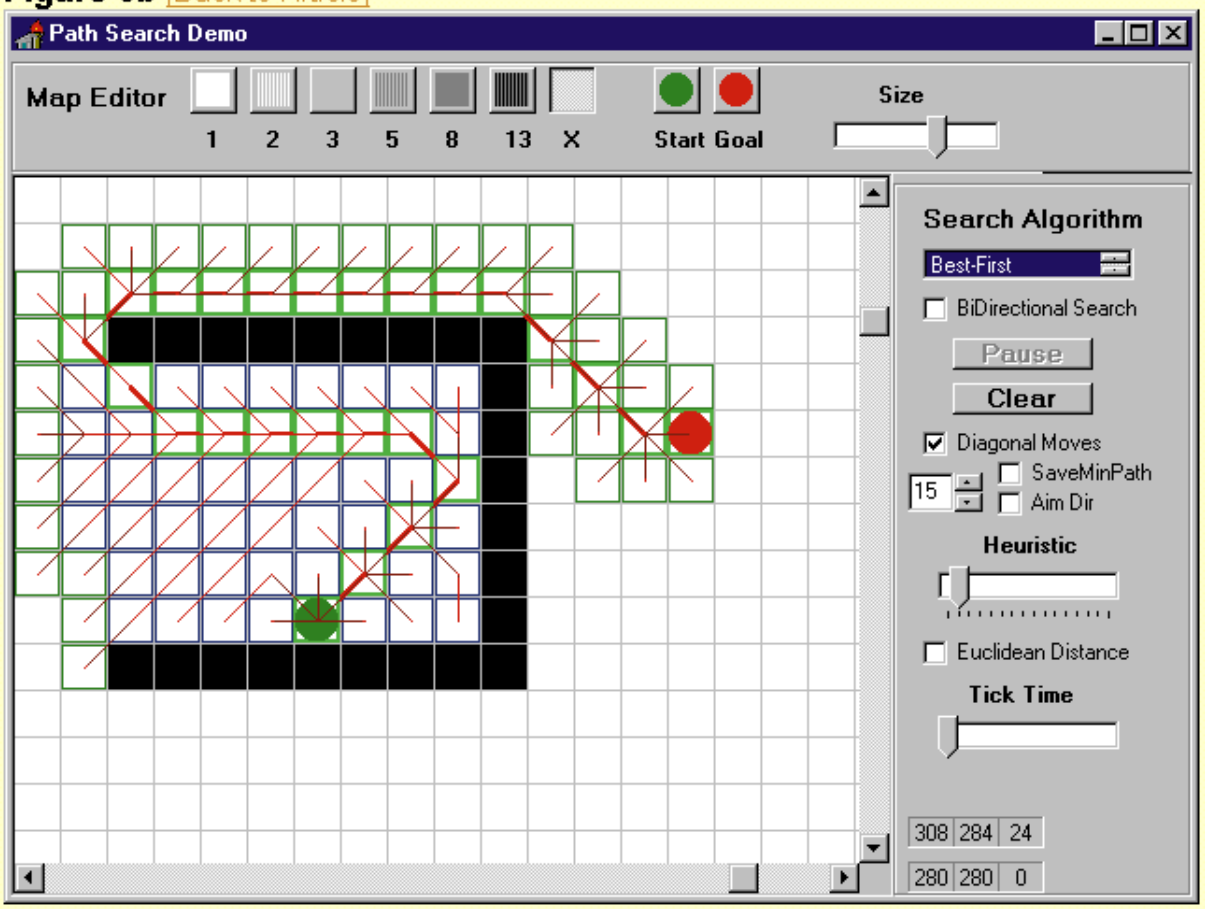


Figure 9a [Back to Article]

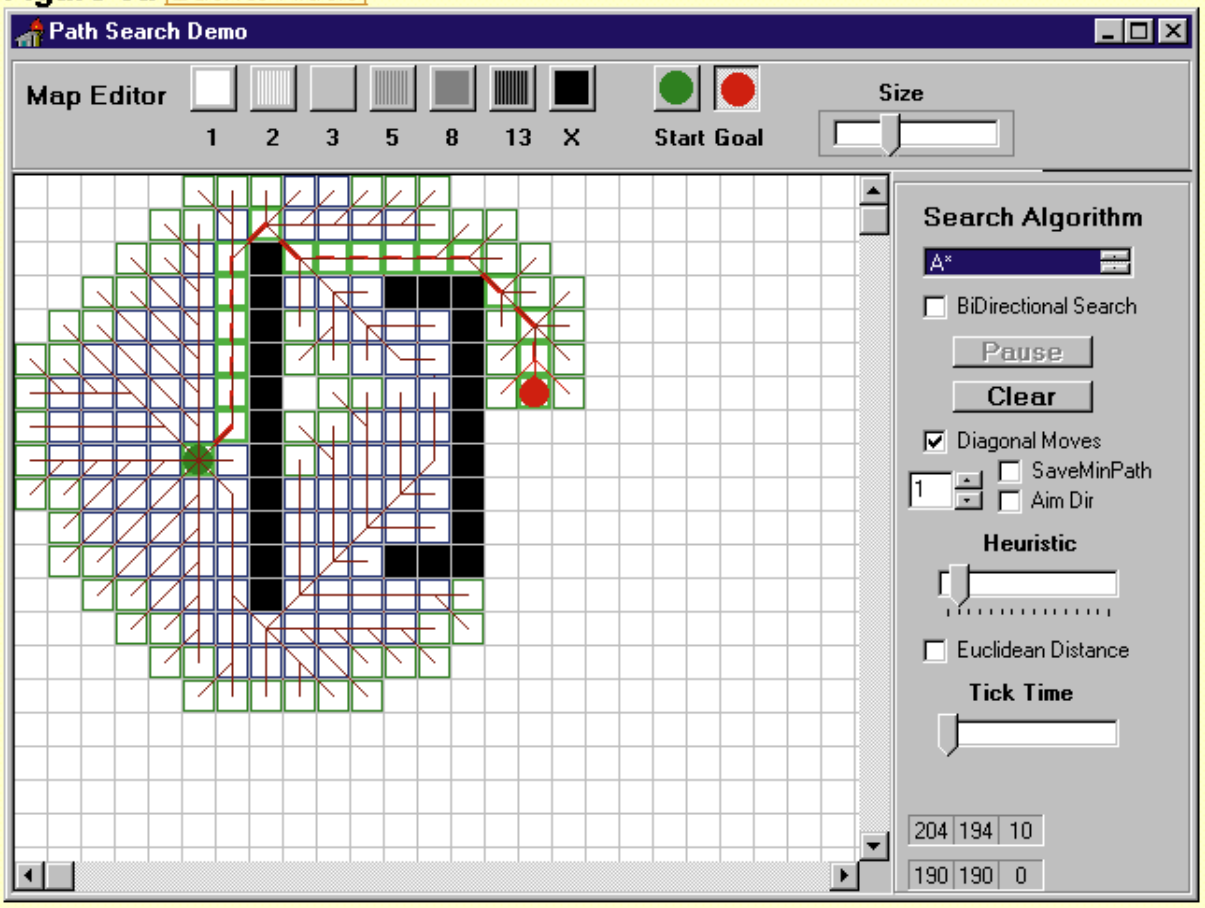


Figure 9b [Back to Article]

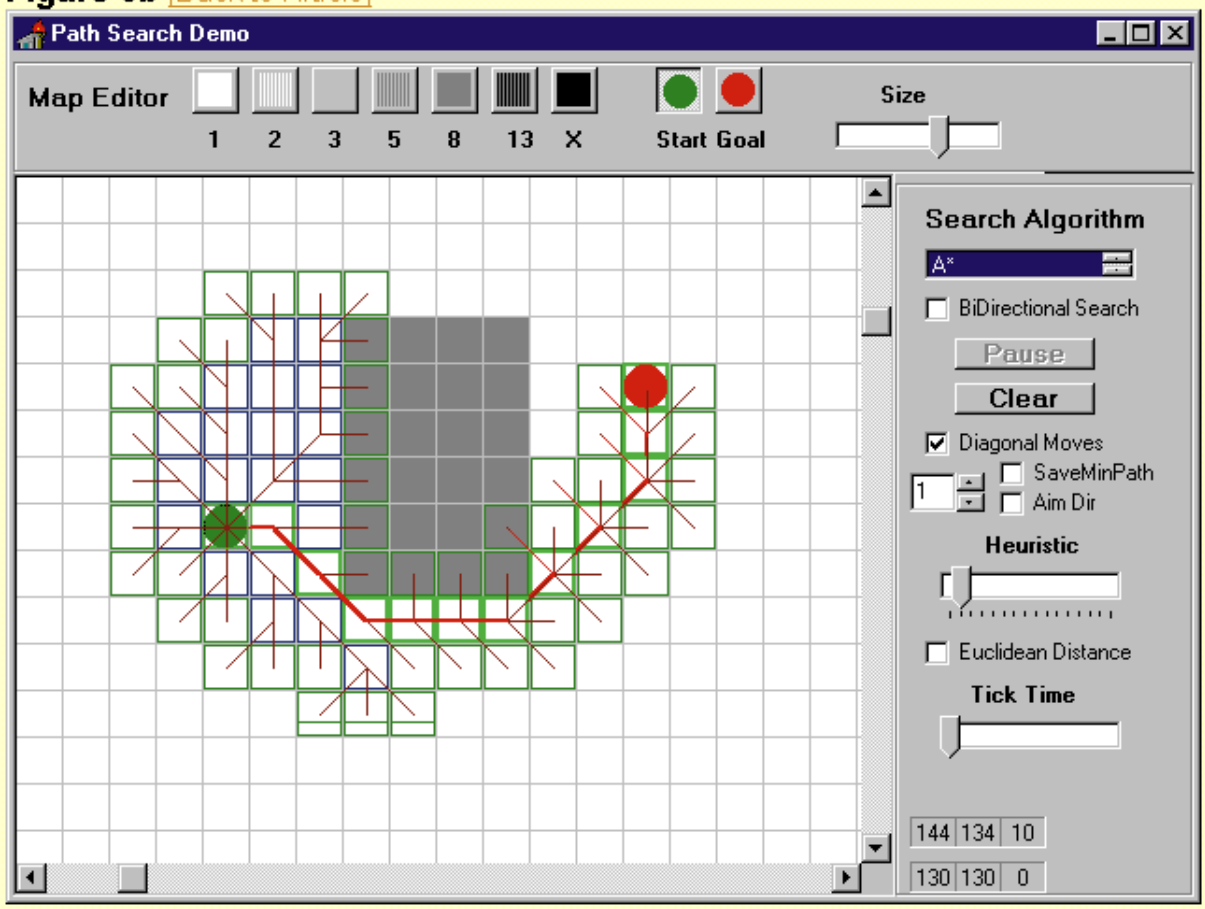


Figure 9c [Back to Article]

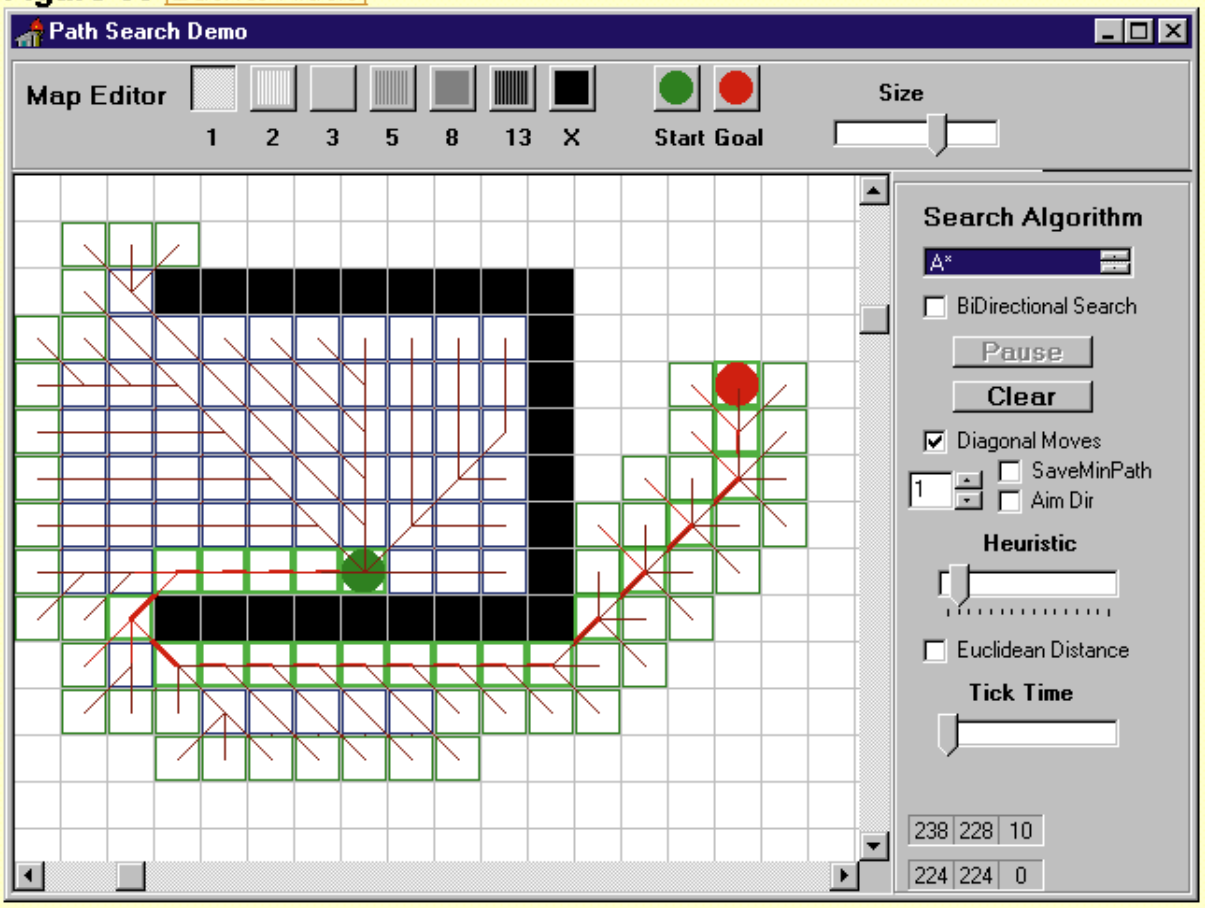


Figure 10a [Back to Article]

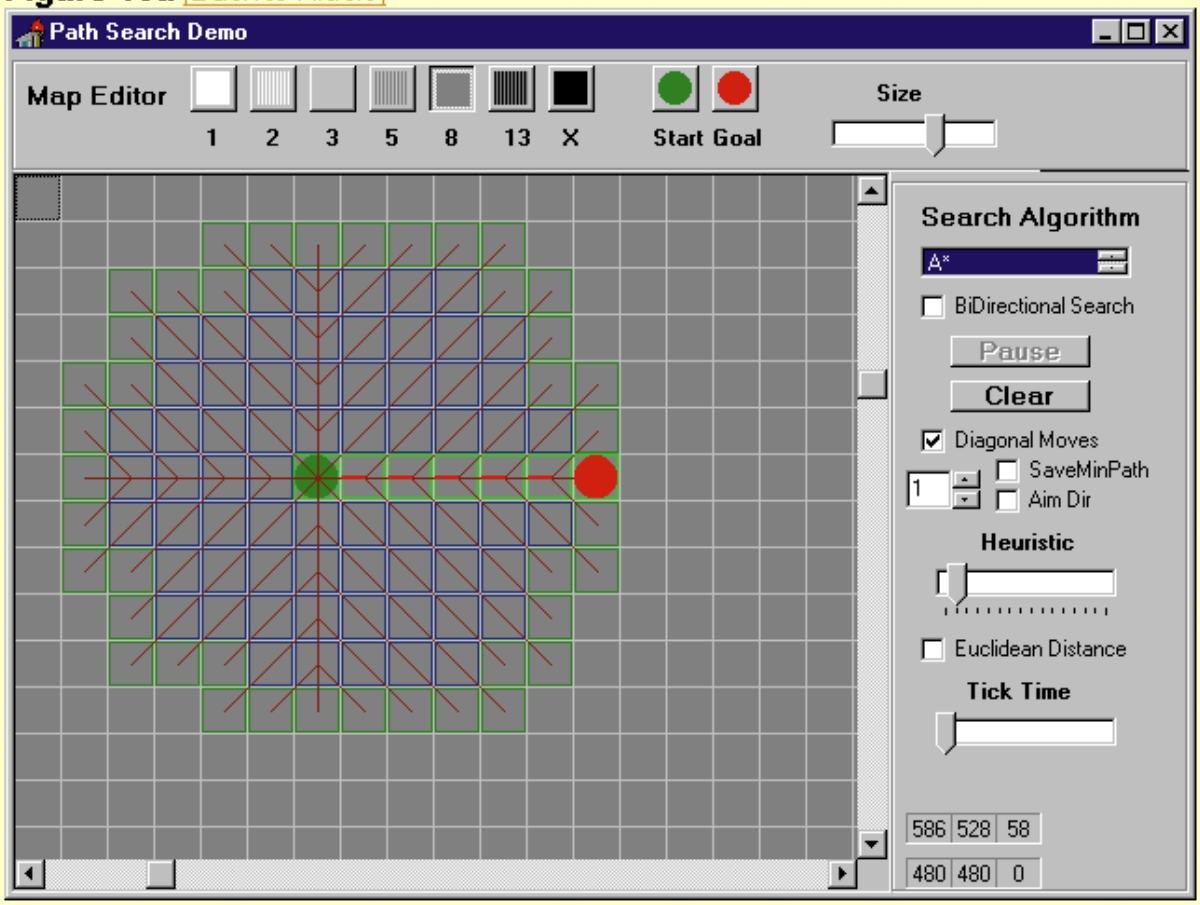
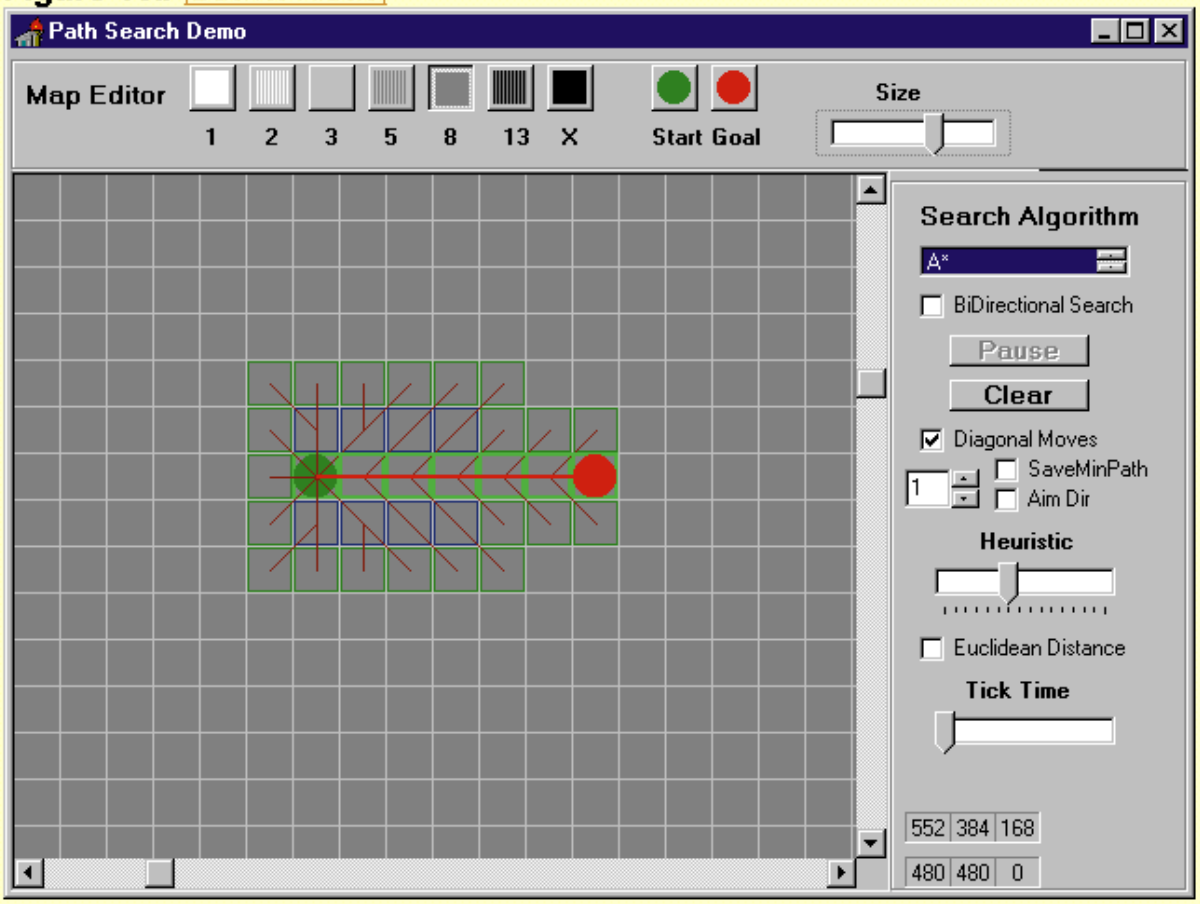


Figure 10b [Back to Article]



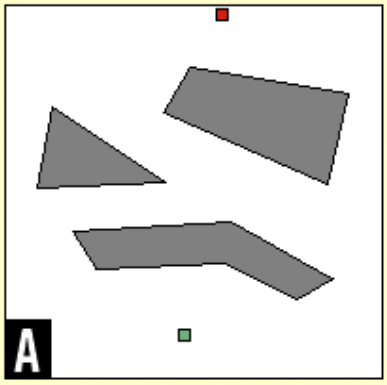


Figure 11a

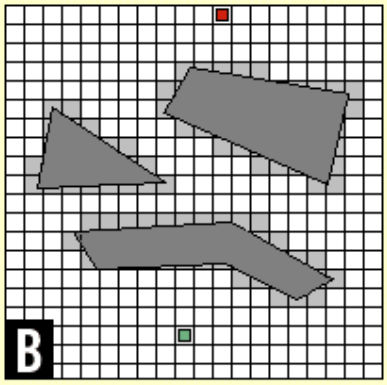


Figure 11b

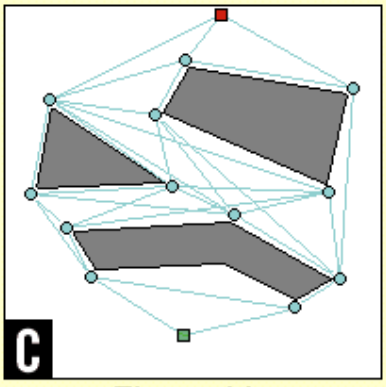


Figure 11c

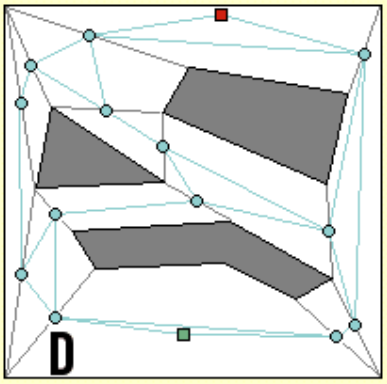


Figure 11d

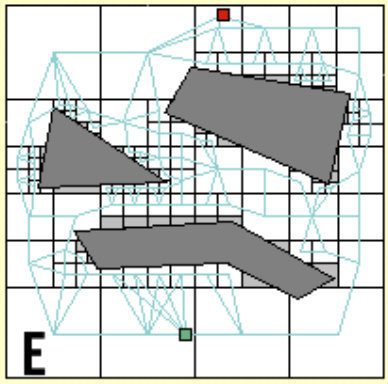


Figure 11e

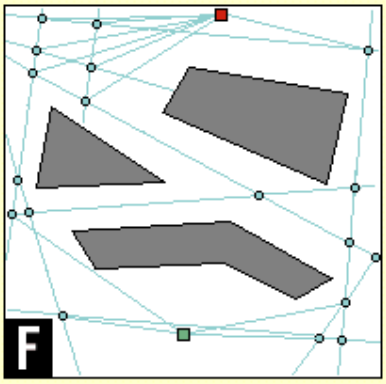


Figure 11f

Listing 1

```
queue Open
BreadthFirstSearch
   node   n, n', s
      s.parent = null   // s is a node for the start
      push s on Open
      while Open is not empty
         pop node n from Open
         if n is a goal node
            construct path
            return success
         for each successor n' of n
            if n' has been visited already,
               continue
            n'.parent = n
            push n' on Open
         return failure  // if no path found
```

Listing 2

```
priority queue Open

DijkstraSearch
   node   n, n', s
      s.cost = 0
      s.parent = null   // s is a node for the start
      push s on Open
      while Open is not empty
         pop node n from Open  // n has lowest cost in
         if n is a goal node
            construct path
            return success
         for each successor n' of n
            newcost = n.cost + cost(n,n')
            if n' is in Open
             and n'.cost < = newcost
               continue
            n'.parent = n
            n'.cost = newcost
            if n' is not yet in Open
               push n' on Open
       return failure // if no path found
```

Listing 3

```
priorityqueue Open
list Closed

AStarSearch
   s.g = 0  // s is the start node
   s.h = GoalDistEstimate( s )
   s.f = s.g + s.h
   s.parent = null
   push s on Open
   while Open is not empty
      pop node n from Open  // n has the lowest f
      if n is a goal node
         construct path
         return success
      for each successor n' of n
         newg = n.g + cost(n,n')
         if n' is in Open or Closed,
          and n'.g < = newg
               skip
         n'.parent = n
         n'.g = newg
         n'.h = GoalDistEstimate( n' )
         n'.f = n'.g + n'.h
         if n' is in Closed
            remove it from Closed
         if n' is not yet in Open
            push n' on Open
      push n onto Closed
   return failure // if no path found
```