# Animation done by a simple text file

Emma @Caligariforum / Moersner
January 2007

In the series of my tutorials this one now is supposed to open the door to the basics for getting data from the outside world that can control the objects of a scenery.

But let us start at the begining which was when I saw an entry in the Caligari forum from Norm http://forums1.caligari.com/truespace/showthread.php?t=235

It was dealing about reading data from a textfile and putting the lines together. Then I found a month ago an old picture in a newspaper from 1896 about a chess game which was playd between England and the USA over the Atlantic cable. Just perfect to be used for a tutorial to show how to get data from an external textfile and use it to move the figures of a chess game.

H. N. Pillsbury.
J. H. Blackburne.
Internationales Kabelschachspiel zwiſchen England und Amerika.

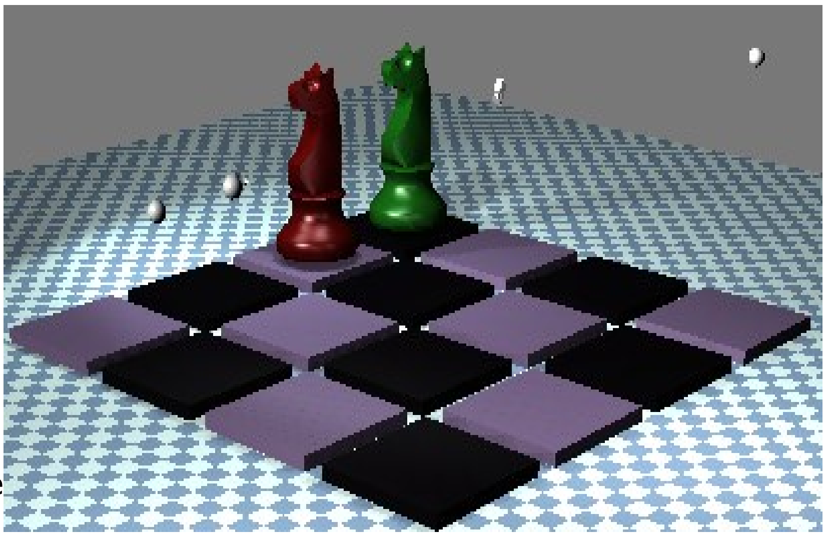
But that is not all, such a script could be used to play any chess game that you have a list of moves from. Perfect for studys, nice to enjoy, extendable by getting animated people around the scenery, adding intelligence for real matches and so on..., your time and fantasy is the limit, never a program.

As ususal we first define the problem, followed by defining the way it should be done and finally the description of the script code for learning, hopefully giving you ideas for lots of innovations. To keep things simple we start with two figures moved on a 4*4 field. This can later be extended to the way chess figures get moved where you usually use the coordinates like *a1* to *a2, d2* to *d4, .....*

- <u>All this is done in **Player** mode !</u> -

**What we want:**
We want a tiny chess board to move figures controlled by data read from a simple text file.

**How should it be done:**
The text file contains a number of lines. Each line contains the fields that tell which figure should be moved how many steps in which direction. For the first example to start with the fields we have in each line are:



- name of figure
- direction (back, forth, left, right)
- steps to move

**We also need**

- a play field arranged 4 * 4
- two figures, named *Figure*1 (or 2)
- delay of movements so they can be seen

As already said in the tutorials before, a good documentation inside the script is first duty. First new thing we will learn now is the access of a text file as Norm described it in the above mentioned link. *ActivXObjekt* is the main little helper for this.

```
            fs = new ActiveXObject("Scripting.FileSystemObject");
```

This is the key that opens the way to it. As you can see you simply use the *ActiveXObject* for this. **fs** is our first key to get *ActiveXObject* file access now.

```
        f = fs.GetFile("G:\\Programme\\truespace71\\scenes\\chess\\location.txt");

        connToFile = f.OpenAsTextStream( forReading, -2);
```

Next **f** becomes our pointing key to the text file. This is in this example hardcoded in the script, means you have to change it to the path you use for your own text file before running the script !

**connToFile** gives now the order that the textfile is to be opened for **Read mode** on a text stream, from first till last byte of data.

```
  // --- read data line by line from the text file into the array rline[]
  var count = 0;
  while( !connToFile.AtEndOfStream )
  {
    rline[count] = connToFile.ReadLine();
    count++;
  }
  connToFile.Close();
```

This conditional loop, a *while* construct, reads now line by line of our data from the text stream. The *while* condition is `!connToFile.AtEndOfStream` which means run this loop until you have gotten the last byte of the data stream. Then you are NOT connected to the file anymore because you now have passed by the END OF DATA STREAM ( **!** stays for NOT condition).

So the array *rline[ ]* will collect the data line by line, where each line should contain *name*, *direction* and *steps*. So far we only worked in the other tutorials with the *for* **{ }** loop construct and the *while* loop is new.

**NOTE:** You have to be very precise with the *while* because if the condition never hits the end you will get an endless loop which you can only kill by killing the whole program with the task manager **!**

After getting last line of data from the text file the connection is closed, data is now in our array *rline [ ]*. This is also an important thing you should remember: close a file as soon as you are done with it so you will not get later on in a file lock situation.

What we have so far is the code to

- get access to a text file

- open the file for read mode only

- reading file data from first to last byte, line by line, into an array

Next to do now is access to the fields in each line which contain *name*, *move* direction and *steps* to go. This is done by separating the fields by a special character. different from all possible data characters. For this sample a ", " is used, but you can of course take any other as long as there is no conflict with data characters.

What follows now is beginning of the meat from our script,

- getting the control data from our text lines
- and turning them into action.

The loop counts from first till last (*rline.length*) line of data. Some calculation is done already ahead, like

- counting the number of lines in *aIndex*,
- splitting apart the fields of the line with *msgB*
- and finally counting the number of fields in that line with *aElements*.

```
var aIndex    = rline.length;          // number of lines
var msgA      = rline[0];              // get first line content
var msgB      = msgA.split(",");       // splitting line data into fields
var aElements = msgB.length;           // number of fields

for(i = 0; i < rline.length; i++)      // running first till last line
{  msg        = rline[i];              // assign single line from rline array.
   SubMsg     = msg.split(",");        // split the line into elements.

   for (a = 0; a < SubMsg.length; a++) // loop through elements per line
   {  if (SubMsg[a] == " ")            //  if element empty = error = stop loop
      {  break; }
      if (a == 0) { movob  = "Figur"+SubMsg[a];}   // string,  Objectname
      if (a == 1) { direct = SubMsg[a];}           // char,  direction to move
      if (a == 2) { schrit = parseInt(SubMsg[a]);} // int,   how many steps
   }
```

So we have one loop now working one line after another and a second loop inside which fills the variables for object name, direction and steps.  With those settings we now can initiate the action to be done. Could be solved several ways, here we use simply the ***if*** command. First we construct the objects name with ***tmp*** and then follows the movement into one of the directions, ***tx*** or ***ty***. The length of the movement depends on how many *steps* are given. The example playfield needs an increment/decrement of **7** and so a simple multiplication by the *step* value for corresponding x/y coordinate will do it.

```
   tmp = ownrnam.concat("/"+movob);
   if (direct == "v")
   { b = Space.NodeMatrixElement(tmp,'tx');
     Space.NodeMatrixElement(tmp,'tx') = b + (7.0 * schrit); }
   if (direct == "z")
   { b = Space.NodeMatrixElement(tmp,'tx');
     Space.NodeMatrixElement(tmp,'tx') = b - (7.0 * schrit); }
   if (direct == "l")
   { b = Space.NodeMatrixElement(tmp,'ty');
     Space.NodeMatrixElement(tmp,'ty') = b + 7.0 * schrit;   }
   if (direct == "r")
   { b = Space.NodeMatrixElement(tmp,'ty');
     Space.NodeMatrixElement(tmp,'ty') = b - 7.0 * schrit;   }

   // this is for moving with delay of 1 second
   while (mm == nn)
   {  Sekunden  = new Date();    // necessary to update the time
      nn        = Sekunden.getSeconds();
      if (mm < nn)
      {  break;}
   }    // --- End of inner loop ---
   D3DView.Rescan();
   D3DView.Refresh();
   mm = nn;
}       // --- End of outer loop ---
```

What follows is a *while* loop to add some delay to each movement so that we have a chance to see it. Finally *Rescan/Refresh* will update the view of the scenery. This makes shure we can see every step and not only the finished status of all the actions.
With this we already have the framework which moves around the figures named as **Figur1** and **Figur2** from our sample scenerey for this tutorial. The name **Figur** is fixed and in the script we

used **tmp** to construct the corresponding objects name. So all we need in our **txt** file would be the <u>number of the object.</u> Do you see the advantage ? Yes, no problem to add more figures, they only have to be named as "Figur"**x** where **x** can be any readable combination of characters from **a** to **z** /**A** to **Z**/**0** to **9**. Notice the advantage of putting already some flexibility with this into the script. As we used Figur1/Figur2 our **txt** file could look like this:

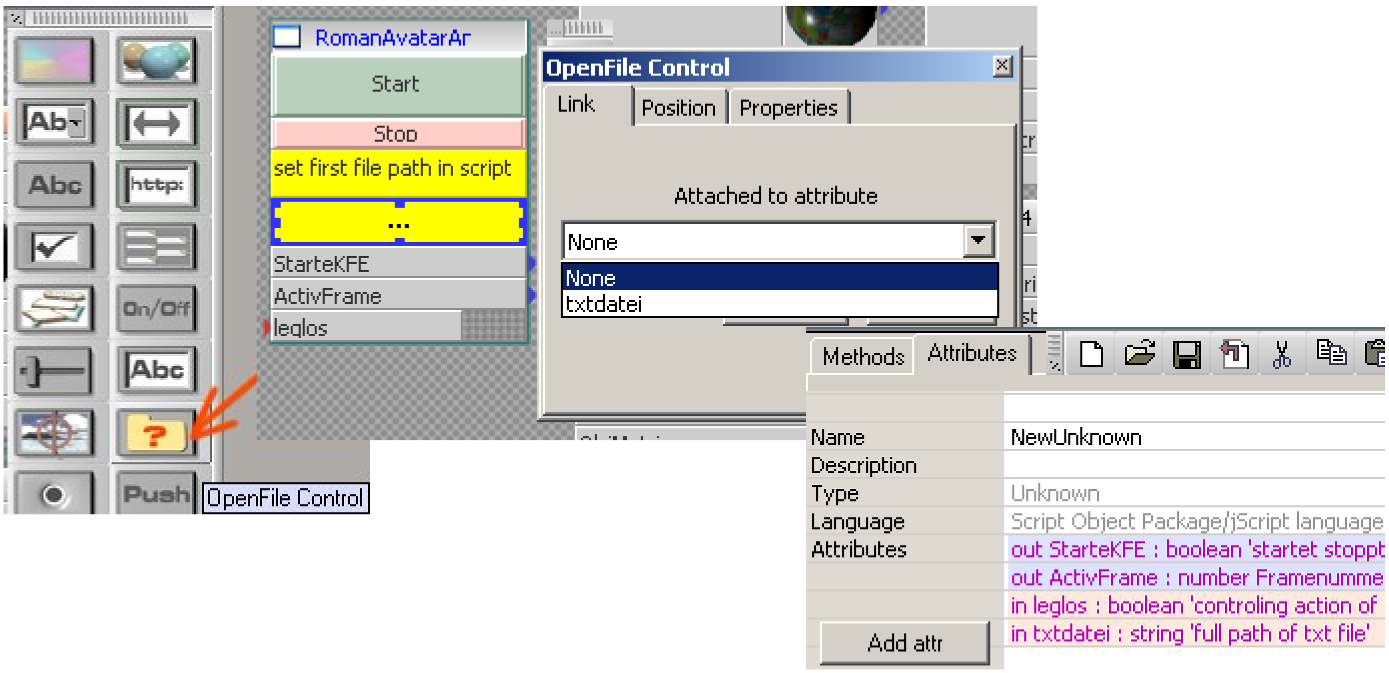
```
1,v,1                                                    green,v,1     or      A,v,1
2,v,2                                                    brown,v,2             M,v,2
1,l,3                                                         .......               ......
2,r,1
2,z,3
```

First entry is the figure number 1 or 2 . Would you prefer to use **A** and **M**, no problem , or rather prefer **green** and **brown** , also no problem. The objects just have to be named this way in your scenery.

Second entry is the direction **v** = forward, **l** = left, **r**= right and **z**= back

Third entry is the number of steps which is used as the multiplicator with 7 to move the figures. You can try now different values to let the figures jump around. Don't hesitate to experiment, but don't forget, alwas keep an eye on **while** coniditons when changing antything of that. In the next tutorial we will add more, like initiating the values for the two figures at start. Momentarily you have to correct them manually ( right mouse-expand on *Matrix* )
Figur1 : tx = -13 ty = -13 tz = 4
Figur2 : tx = -13 ty = -6 tz = 4

We will already take one example from the next part ahead. At the moment you have to change the path of the text file inside the script. You can do this also from the outside the following way.



| | |
|---|---|
| Name | NewUnknown |
| Description | |
| Type | Unknown |
| Language | Script Object Package/jScript language |
| Attributes | out StarteKFE : boolean 'startet stoppt |
| | out ActivFrame : number Framenumme |
| | in leglos : boolean 'controling action of |
| | in txtdatei : string 'full path of txt file' |

Simply add the **OpenFile Control** and **Link** it to **txtdatei**. This variable **txtdatei** must be defined on the *Attributes tab* of the script. The place to use it in the script is in the line:

```
f = fs.GetFile("G:\\Programme\\truespace71\ts\\Szenen\\Schachtest\\location.txt");
```

I'm sure I don't have to tell you how.

So this should be the basics how to use data of a txt file for getting objects into action.

```
function Execute(params)
// Script : positionizer                              Emma at 3d-ts-forum/Caligariforum
// Command - Version                                          January 2007 / M.Moersner
//       26.1.20.55
// mainly based on sample script published by Norm in the Caligari forum
// --> read data from a text file and separate the information to do something with it
// added other functionalitys that members published in Caligari forum
// --> get access to Node elements, get access to time
// extended and changed then to be used for an animation sample with scripting
// --> idea for practical use, writing and documenting code, prepare tutorial like
//
// aIndex      - number of all characters contained in rline[]
// msgA        - used to store temporary one complete line from array rline[]
// aElements   - number of array fields, done with special (no () used!) function:
msgB.length
// msgB[]      - separates msgA into an array, separating with function:
msgA.split("separating sign")
//
// rline[]     - each arrayelement contains 1 line of Data from the textfile
// msg         - stores 1 array element from rline
// SubMsg      - stores
// leglos      - used for start/stop delay between moves
// ------- DATA FILE STRUCTURE: 3 elements per line
// element 1 - object to move, number from 1 to x, added to objects basename
// element 2 - direction of movement: v = forward, z = backward, l = left, r = right
// element 3 - number of movement steps

{
  ownrnam = System.ThisOwner();

  var forReading = 1                // , forWriting = 2, forAppending = 8;
  var Sekunden   = new Date();      // necessary for time/date functions

  rline  = new Array();
  fs     = new ActiveXObject("Scripting.FileSystemObject");

// ==>W A T C H  O U T:the path name below depends on where you placed the .txt file<==
    f = fs.GetFile("G:\\Programme\\truespace71\\ts\\Szenen\\Schachtest\\location.txt");

  connToFile = f.OpenAsTextStream( forReading, -2);

  // --- read data line by line from the text file into the array rline[]
  var count = 0;
  while( !connToFile.AtEndOfStream )
  {
    rline[count] = connToFile.ReadLine();
    count++;
  }
  connToFile.Close();

  // -- all data read from file, now prepare for use and first assign variables
  //     i counts the number of array elements
  // --- a counts the number of elements per line
  var aIndex    = rline.length;
  var msgA      = rline[0];
  var msgB      = msgA.split(",");
  var aElements = msgB.length;
  dataArray     = new Array(aIndex);
  movob         = " ";
  direct        = " ";
  schrit        = 0;
  Sekunden      = new Date();           // necessary for time/date functions
  mm            = Sekunden.getSeconds();
  nn            = mm ;

  for(i = 0; i < rline.length; i++)
  {
    dataArray[i]= new Array(aElements);
    msg         = rline[i];             // get single line from rline array.
```

```
      SubMsg      = msg.split(",");       // split the line into elements.

      for (a = 0; a < SubMsg.length; a++) // loop through elements per line
      {  if (SubMsg[a] == " ")            //  if element empty = error = stop loop
         {  break; }
         if (a == 0) { movob  = "Figur"+SubMsg[a];}   // string,  Objectname
         if (a == 1) { direct = SubMsg[a];}           // char,  direction to move
         if (a == 2) { schrit = parseInt(SubMsg[a]);} // int,  how many steps
      } // --- End inner loop ---

      tmp = ownrnam.concat("/"+movob);
      if (direct == "v")
      { b = Space.NodeMatrixElement(tmp,'tx');
        Space.NodeMatrixElement(tmp,'tx') = b + (7.0 * schrit); }
      if (direct == "z")
      { b = Space.NodeMatrixElement(tmp,'tx');
        Space.NodeMatrixElement(tmp,'tx') = b - (7.0 * schrit); }
      if (direct == "l")
      { b = Space.NodeMatrixElement(tmp,'ty');
        Space.NodeMatrixElement(tmp,'ty') = b + 7.0 * schrit;   }
      if (direct == "r")
      { b = Space.NodeMatrixElement(tmp,'ty');
        Space.NodeMatrixElement(tmp,'ty') = b - 7.0 * schrit;   }

      // this is for moving with delay of 1 second
      while (mm == nn)
      {  Sekunden  = new Date();    // necessary to update the time
         nn        = Sekunden.getSeconds();
         if (mm < nn)
         {  break;}
      }
      D3DView.Rescan();
      D3DView.Refresh();
      mm = nn;
   }// --- End outer loop ----

   // make just a fast jump for fun :-) therfore using milliseconds
   // use tmp since that should still contain last figure used
   //
   b = Space.NodeMatrixElement(tmp,'pitch');
   for (i=1; i<7; i++)
   { b= b + 60;
      Sekunden = new Date();
      mm = Sekunden.getMilliseconds();
      nn = mm;
      while (mm == nn)
      {  Sekunden  = new Date();    // necessary to update the time
         nn        = Sekunden.getMilliseconds();
         nxm = nn -mm;
         if (nxm > 500)
         {  break;}
      }
      Space.NodeMatrixElement(tmp,'pitch') = b;

      D3DView.Rescan();
      D3DView.Refresh();
   }
}
```