Unity Dance 1d Cellular Automata

Ath. Kehagias
`thanasiskehagias@gmail.com`

June 1, 2018

1 Introduction

It is a Unity3d project in which you can set up some agents to perform dance moves when activated by a 1d Cellular Automaton Rule.

It has a *modding* capability in the following sense: the agents' movements are determined by some script which is located in the *StreamingAssets* folder and which you can modify inside the application (i.e., you do not need to open the project in the Unity editor).

It is programmed mainly in Javascript. I have tested it on Unity 2017.3.f3 (it will probably work on any Unity 5.* and higher) and on Win7 (may need serious tweaking to work on iOS, Android etc.).

2 Quick Start

2.1 For the App

Download the project in zipped form (from `GitHub.com` you will get a file `Unity-Dance-CA.zip`) and unzip it wherever. You get a folder named `Unity-Dance-CA`, go inside and then into subfolder `Unity-Dance-CA/Apps` and run `DanceCA.exe`. You will see the dudes doing their thing.

You can do more. Go into the folder `Unity-Dance-CA/Apps/DanceCA_Data/StreamingAssets`, you will see some `*.ogg` sound files and some `*.txt` script files. You can copy here more sound files (they must be in `ogg` format); if you add `MySong.ogg` and change the first line of file `DanceFloor01Start.txt` to

```
MusicFile="MySong.ogg"
```

then when you restart the App it will play your song.

By tweaking the `*.txt` files you can accomplish several other things. To find out more, read the next section.

2.2 For the Project

`Unity-Dance-CA/` is a regular Unity project folder. Open it in the Unity Editor. You can do the usual Unity things which, presumably, you know how to do if you have read this far. I will only mention here the *modding* capability.

The basic idea behind this project is that you can create some `*.txt` *script files* and place them in the StreamingAssets folder; then, after you build your executable, you can go into *its* StreamingAssets folder and modify the scripts so that you get different behaviors. For example, the current 1d CA rule which activates the dancers, is defined in the `DanceFloor01Start.txt` file which reads as follows:

```
MusicFile="402018__frankum__monster-beats-edm-music.ogg";
CARule[0]=1;
CARule[1]=0;
CARule[2]=0;
CARule[3]=1;
CARule[4]=0;
CARule[5]=1;
CARule[6]=0;
CARule[7]=1;
NSites=20;
REST=120;
TIME=0;
```

So the first line defines which music file will be played, the final lines define some simulation parameters and lines 2-9 define the 1d CA rule[1]. You can change them in the `Assets/StreamingAssets` folder (and when building they will be transferred to the App) but you can also change them directly in the `Apps/.../StreamingAssets` folder (after the build).

This modding capability is obtained by using the JavaScript `eval(string)` function, where `string` can be any sequence of valid JavaScript commands (well, *almost* any sequence; it is a little more complicated). You can get ideas about what is possible to achieve by looking at the existing `*.txt` files and also by reading the next section.

3 Details

The structure of the project is such that for every `*.js` script file, there exists a `*Start.txt` and a `*Update.txt` file. The idea is that the `*.js` sets some basics up and then: (i) in its `Start()` function runs the `*Start.txt` script; (i) in its `Update()` function runs the `*Update.txt` script. So the main action is determined by the `*.txt` files. Here is an example. There is a `Camera01.js` script which goes like this.

```
var StartName:String;
var StartCode:String;
var UpdateName:String;
var UpdateCode:String;
var Target:GameObject;
var R:float;
var w:float;
var TIME:int;
var CamState:int;
var minFov: float = 10f;
var maxFov: float = 90f;
var sensi2: float = 10f;
var fov: float;
function Start ()
{
   StartName="Camera01Start.txt";
   StartCode=ReadScript(StartName);
   UpdateName="Camera01Update.txt";
   UpdateCode=ReadScript(UpdateName);
   eval(StartCode);
}
function Update ()
{
   eval(UpdateCode);
}
```

[1] This gives the activation level 0 or 1 for a site of the CA, depending on the activation levels of itself and its two immediate neighbors in the previous time step. For more details see `https://en.wikipedia.org/wiki/Elementary_cellular_automaton`.

```
function ReadScript(ScriptName:String):String
{
   var ScriptCode:String;
   var fn=Application.dataPath + "/StreamingAssets/"+ScriptName;
   if(System.IO.File.Exists(fn))
   {
     var sr0 = new StreamReader(fn);
     ScriptCode = sr0.ReadToEnd();
     sr0.Close();
   }
   return ScriptCode;
}
```

It starts with some variable declarations. Note the `StartCode` variable: this is where the text of `Camera01Start.txt` will be stored; similarly, the `UpdateCode` is where the text of `Camera01Update.txt` will be stored. The final variable declarations are about camera behavior parameters.

The a `Start()` function of `Camera01.js` basically loads `StartCode` and `UpdateCode` (using the function `ReadAScript(FileName)`) and then executes `StartCode` (using the function `eval(StartCode)`). The contents of `Camera01Start.txt` are the following.

```
TIME=0;
fov=3f;
CamState=1;
```

So basically it sets up some camera parameters. These are used by the contents of `Camera01Update.txt`, which are the following.

```
if(TIME%SwitchTime==0)  CamState=Random.Range(1,4);
if (Input.GetKeyDown (KeyCode.F1)) CamState=1;
if (Input.GetKeyDown (KeyCode.F2)) CamState=2;
if (Input.GetKeyDown (KeyCode.F3)) CamState=3;
if(CamState==1)
{
    transform.position=Vector3(0,2*H1,0);
}
if(CamState==2)
{
    transform.position=Vector3(R0*Mathf.Sqrt(2)/2,H1,-R0*Mathf.Sqrt(2)/2);
}
if(CamState==3)
{
    transform.position=Vector3(R0*cos(w1*Time.time),1,R0*sin(w2*Time.time));
}
transform.LookAt(Target.transform);
```

So it implements three camera behaviors, activated by keys `F1, F2, F3` or in a random way at fixed times.

The point is that if you want different camera behaviors, you will go and change the contents of `Camera01Update.txt`. For example, adding the following lines

```
if(CamState==4)
{
    transform.position=Vector3(0,H1,0);
}
```

places the camera at the center of the dancing floor (do not forget to add a keyboard check for `F4`, to set `CamState` to 4).

You can do similar things with the DanceFloor scripts. To get into some detail, `DanceFloor01Update.txt` contains the following.

```
TIME=TIME+1;
if(TIME==2)
{
 var R1=6f;
 Oracle.transform.position.x=0;
 Oracle.transform.position.z=0;
 for(n=0;n<NSites;n++)
 {
  Dancers[n].transform.position.x=R1*Mathf.Cos(n*6.28/NSites);
  Dancers[n].transform.position.z=R1*Mathf.Sin(n*6.28/NSites);
  Dancers[n].transform.LookAt(Oracle.transform);
 }
}
for(n=0;n<NSites;n++)
{
  AniConts[n].SetInteger("TargState",-1);
}
if(TIME==REST)
{
 TIME=0;
 for(n=0;n<NSites;n++)
 {
  ValuesOld[n]=Values[n];
 }
 for(n=0;n<NSites;n++)
 {
  m=n-1; if(m<0) m=n+NSites; xm=ValuesOld[m];
  m=n; x0=ValuesOld[m];
  m=n+1; if(m>NSites-1) m=n-NSites+1; xp=ValuesOld[m];

  if(xm==1 && x0==1 && xp==1) Values[n]=CARule[0];
  if(xm==1 && x0==1 && xp==0) Values[n]=CARule[1];
  if(xm==1 && x0==0 && xp==1) Values[n]=CARule[2];
  if(xm==1 && x0==0 && xp==0) Values[n]=CARule[3];
  if(xm==0 && x0==1 && xp==1) Values[n]=CARule[4];
  if(xm==0 && x0==1 && xp==0) Values[n]=CARule[5];
  if(xm==0 && x0==0 && xp==1) Values[n]=CARule[6];
  if(xm==0 && x0==0 && xp==0) Values[n]=CARule[7];

 }
 m=Random.Range(1,4);{}
 for(n=0;n<NSites;n++)
 {
  AniConts[n].SetInteger("TargState",0); if( Values[n]) AniConts[n].SetInteger("TargState",m);
  if( Values[n]) Sites[n].transform.localScale=Vector3(1.0,0.1,1.0);
  if(!Values[n]) Sites[n].transform.localScale=Vector3(0.1,0.1,0.1);
 }
}
```

Here is a partial explanation. The first loop (executed only when `TIME==2`) positions a set of dancers along the perimeter of a circle (as you can see when you run the scene). Then there is a loop which is only executed every `REST` time steps, and which updates the activation values `Values[n]` of each site; if a value is active (`Values[n]==1`) the parameter `TargState` of the respective Animation Controller `AniConts[n]` is set to a value which runs a dancing animation; otherwise (`Values[n]==0`) it runs an idle animation.

In short: the CA activations are periodically updated, and when a site is active (resp. inactive) the corresponding dancer dances (resp. idles). The point is that, if you want a different dancing behavior, you will go and change the definition of the CA rule in `DanceFloor01Start.txt`.

Generally you can write any valid JavaScript code in the `*.txt` files. Of course, the code will be valid only if it operates on existing variables, so you need to look at the beginning of the `*.js` files to see which variables have been declared.

I must admit that my code is a *Hack of Hacks* and I am sure the job could be done much better; so feel free to rectify and extend the code as you please. Happy hacking!

4 Assets Used

Hearty thanks to the providers of following.

1. Mixamo figures and animations (free from `https://www.mixamo.com/`.
2. Shader: MkGlow (free from Unity Asset Store `https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/mk-glow-free-28044`).
3. Tower Mesh: from Sketchup's 3dWarehouse (free at `https://3dwarehouse.sketchup.com/?hl=en`).
4. Script BGLoader.js for loading sound files at runtime. I found it on the net but I cannot remember a good link to it. Sorry!
5. Music file *Monster Beats* by Frankum at `https://freesound.org/people/frankum/sounds/402018/` (from `https://freesound.org/`).