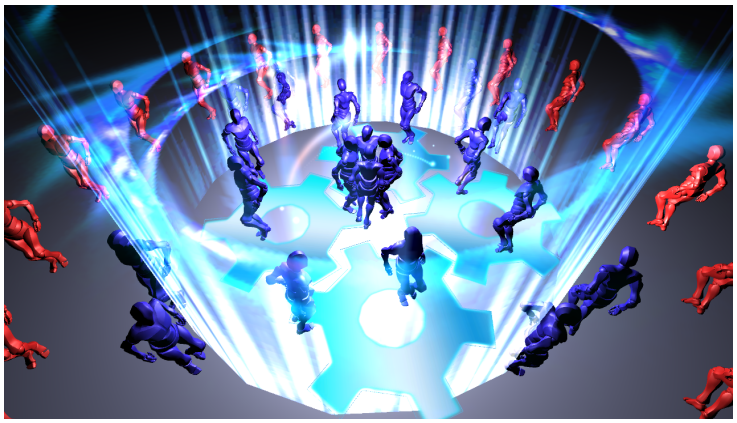
Unity Dance Curves

Ath. Kehagias
thanasiskehagias@gmail.com

May 29, 2018



1 Introduction

It is a Unity3d project in which you can set up some agents to follow mathematical curves and perform dance moves.

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.). The main files are:

1. `DanceCurves.zip` (the project files; feel free to modify as you please);
2. `DanceCurvesApp.zip` (the compiled executable);
3. `DanceCurves.pdf` (documentation – this file).

2 Quick Start

2.1 For the App

Unzip `DanceCurvesApp.zip` to wherever. You get a folder named `DanceCurvesApp`; go into that and run `DanceCurves.exe`. You will see the dudes doing their thing.

You can do more. Go into the folder `DanceCurvesApp/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 `Control01Start.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

Unzip `DanceCurves.zip` to a `DanceCurves` folder; it 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 curves along which the agents dance are defined in the `Dancer00AUpdate.txt` file as follows:

```
r=R0*cos(K*(w*Time.time+Phi));
transform.position.x=r*cos(w*Time.time+Phi);
transform.position.y=0;
transform.position.z=r*sin(w*Time.time+Phi);
```

You can change these lines and use a different curve. You can change them in the Project StreamingAssets folder (and when building they will be transferred to the App) but you can also change them directly in the App 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.m

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.

```
import System.IO;
var Target :  GameObject;
var TIME:int;
var SwTime:int;
var StartName:String;
var StartCode:String;
var UpdateName:String;
var UpdateCode:String;
var H1:float;
var R0:float;
var w1:float;
var w2:float;
var CamState:int;
var SwitchTime:int;
function Start ()
{
    TIME=0;
    StartName="Camera01Start.txt";
    StartCode=ReadScript(StartName);
    UpdateName="Camera01Update.txt";
    UpdateCode=ReadScript(UpdateName);
    eval(StartCode);
}
function Update ()
{
    TIME=TIME+1;
    eval(UpdateCode);
}
```

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.

```
H1=5;
R0=5;
w1=0.25;
w2=0.35;
CamState=3;
SwitchTime=500;
```

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 Control and Dancer sccripts. There actually are two teams, the Blue Dancers and the Red Dancers, controlled by `Dancer00A.js` and `Dancer00B.js` respectively. The contents of `Dancer00AUpdate.txt` are the following.

```
r=R0*cos(2*(w*Time.time+Phi));
transform.position.x=r*cos(w*Time.time+Phi);
transform.position.y=0;
transform.position.z=r*sin(w*Time.time+Phi);
```

These implement the *polar equation* of a *four-leaved* rose and this is the trajectory of the Blue Dancers. If you want them to perform a *three-leaved* rose, just change the first line to

```
r=R0*cos(3*(w*Time.time+Phi));
```

Generally you can write any valid JavaScript code in the `*.txt` files (note also the above use of `cos, sin` etc. instead of `Mathf.Cos, Mathf.Sin` etc.; these are "shortcut-notations" I have defined for the same functions). 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. Particles: Teleporter (free from Unity Asset Store `https://assetstore.unity.com/packages/vfx/particles/teleporter-25645`).
3. Particles: Particle Ribbon (free from Unity Asset Store `https://assetstore.unity.com/packages/vfx/particles/spells/particle-ribbon-42866`).
4. Script for JavaScript-to-CSharp and CSharp-to-JavaScript `http://www.41post.com/1935/programming/unity3d-js-cs-or-cs-js-access`.
5. 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!
6. Music file *Monster Beats* by Frankum at `https://freesound.org/people/frankum/sounds/402018/` (from `https://freesound.org/`).