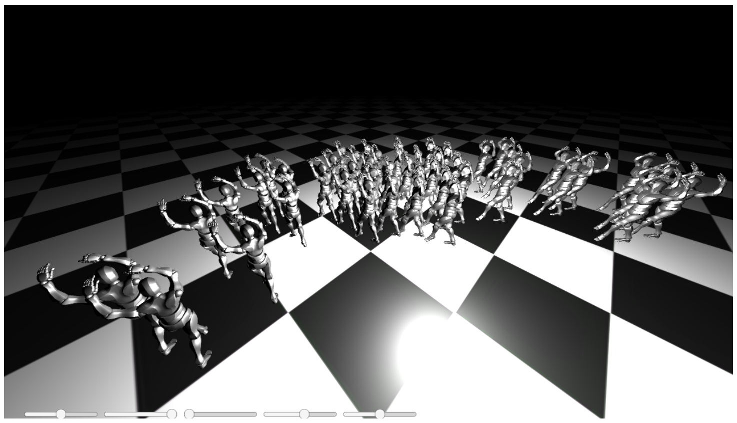
Unity Dance Flocking

Ath. Kehagias
thanasiskehagias@gmail.com

June 9, 2018



1 Introduction

It is a Unity3d project in agents perform a *flocking dance*, i.e., they move in space following Craig Reynolds' *Boid* flocking (or swarming rules).

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-Flocking.zip`) and unzip it wherever. You get a folder named `Unity-Dance-Flocking`, go inside and then into subfolder `Unity-Dance-Flocking/Apps` and run `FlockingDance.exe`. You will see the dudes doing their thing.

You can do more. Go into the folder `Unity-Dance-Flocking/Apps/FlockingDance_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 `MusicPlayerStart.txt` to

```
targetFilename="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-Flocking` 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 `MyControllerUpdate.txt` file which starts as follows:

```
DroneParams.neighborRadius  =100.0f*(3.0f+Mathf.Sin(0.200f*Time.time));
DroneParams.separationWeight=3.50f*(1.0f+Mathf.Sin(0.120f*Time.time));
DroneParams.alignmentWeight =3.50f*(1.0f+Mathf.Sin(0.160f*Time.time));
DroneParams.cohesionWeight  =4.20f*(0.0f+Mathf.Sin(0.240f*Time.time)+0.20f);
/*
DroneParams.neighborRadius  =100f;
DroneParams.separationWeight=3f;
DroneParams.alignmentWeight =3f;
DroneParams.cohesionWeight  =4f;
*/
```

These lines define some of the flocking parameters. So if you comment the first four lines and uncomment the next four ones, you will have different flocking behavior. Try it! (Note that the actual flocking code is not mine, I borrowed it from *robu3*'s *swarming-unity* project found at `https://github.com/robu3/swarming-unity`. Also, the original flocking algorithm was invented by Craig Reynolds, see `https://en.wikipedia.org/wiki/Boids`.)

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 `Camera02.js` script which goes like this.

```
import System.IO;
var Target : GameObject;
var StartName:String;
var StartCode:String;
var UpdateName:String;
var UpdateCode:String;
var TIME:int;
var CamState:int;
var horizontalSpeed : float = 40.0;
var verticalSpeed : float = 40.0;
var minFov: float = 10f;
var maxFov: float = 90f;
var sensi1: float = 1f;
var sensi2: float = 10f;
var fov: float;
function Start ()
{
    TIME=0;
    StartName="Camera02Start.txt";
    StartCode=ReadScript(StartName);
    UpdateName="Camera02Update.txt";
    UpdateCode=ReadScript(UpdateName);
    eval(StartCode);
}
function Update ()
{
    eval(UpdateCode);
}
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 `Camera02Start.txt` will be stored; similarly, the `UpdateCode` is where the text of `Camera02Update.txt` will be stored. The final variable declarations are about camera behavior parameters.

The `Start()` function of `Camera02.js` basically loads `StartCode` and `UpdateCode` (using the function `ReadAScript(FileName)`) and then executes `StartCode` (using the function `eval(StartCode)`). The contents of `Camera02Update.txt` are the following.

```
    if (Input.GetKeyDown (KeyCode.F1)) CamState=1;
    if (Input.GetKeyDown (KeyCode.F2)) CamState=2;
    if (Input.GetKeyDown (KeyCode.F3)) CamState=3;
    TIME=TIME+1;
    fov = Camera.main.fieldOfView;
    fov -= Input.GetAxis("Mouse ScrollWheel") * sensi2;
    if(CamState==1)
    {
        fov=17+73*Mathf.Abs(Mathf.Sin(0.0015*TIME));
        transform.position=Target.transform.position+Vector3(30,30,30);
        transform.LookAt(Target.transform);
    }
    if(CamState==2)
    {
        transform.LookAt(Target.transform);
    }
    if(CamState==3)
    {
        if (Input.GetKey(KeyCode.LeftArrow)) transform.eulerAngles.y-=0.5;
        if (Input.GetKey(KeyCode.RightArrow)) transform.eulerAngles.y+=0.5;
        if (Input.GetKey(KeyCode.UpArrow)) transform.eulerAngles.x-=0.5;
        if (Input.GetKey(KeyCode.DownArrow)) transform.eulerAngles.x+=0.5;
        if (Input.GetKey(KeyCode.PageUp)) fov+=0.5;
        if (Input.GetKey(KeyCode.PageDown)) fov-=0.5;
     }
    fov = Mathf.Clamp(fov, minFov, maxFov);
    Camera.main.fieldOfView = fov;
```

So it implements three camera behaviors, activated by keys `F1, F2, F3`.

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).

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. Flocking code from `https://github.com/robu3/swarming-unity`.
3. 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!
4. Music file 150413_Weird_Electro from `https://freesound.org/`.