Dancing with Friends and Enemies

Jesse Bettencourt
2016/01/26

Acknowledgment

This system was introduced to me by a post on the Wolfram Community forum **Dancing with friends and enemies: boids' swarm intelligence** posted by Simon Woods. In his post, Woods describes the rules for a particle system in which each particle is a dancer with dancing rules. Woods also provides extremely concise *Mathematica* code, totaling only 8 lines, that fully describe and demonstrate this dance. I've written a far less concise implementation of this system, for personal understanding, practice with *Mathematica*, and also to allow me to tinker with the dancing rules to explore new behaviour. Also, it's worth mentioning that this particle system, like all dances, is best appreciated with some music.

The Dance

We are investigating a dance with n dancers. At the beginning of the dance, the dancers are scattered randomly around the dancefloor. Each dancer chooses at random a friend and an enemy among their fellow dancers.

At each 'beat' of the dance, each dancer moves according to the following steps:

1. A step towards the center of the dancefloor.
2. A step towards their friend.
3. A step away from their enemy.

Further, at random intervals a dancer will re-choose their friend and enemy.

```
(*The Number of Dancers*)
n = 1000;


(*Randomly places dancers on the floor and chooses the friends and enemies*)
r := RandomInteger[{1, n}, n];
InitializeDance[dim_: 2] := Module[{x, friends, enemies},
  x = RandomReal[{-1, 1}, {n, dim}];
  friends = r;
  enemies = r;
  {x, friends, enemies}
 ]
{x, f, e} = InitializeDance[];
```

```
(*In Woods' initial code, he had the random event for choosing a new partners
 given by just a univariate 1/10
 probability. I decided to sample a PoissonDistribution
 with mean of 3 (chosen arbitrarily). And if
 the sample is above a threshold parameter
 then the partners switch. *)
NewPartnerQ[threshold_] := RandomVariate[PoissonDistribution[3]] ≥ threshold;
ReChoose[type_, i_] := ReplacePart[type, i -> RandomInteger[{1, n}]];
ChooseNewPartner[friends_, enemies_, thresh_: 4, numnew_: 1] :=
 Module[{i}, If[NewPartnerQ[thresh],
   i = RandomInteger[{1, n}];
   {ReChoose[friends, i], ReChoose[enemies, i]},
   {friends, enemies}]]

(*Here we have the functions which specify the steps of the dance. I could have
 simplified this probably to one function but kept the separate functions because
 of both legacy later and also to allow me to add new steps modularly.*)
DirectionVector[from_, to_] := Normalize /@ (to - from)
StepToCenter[from_, amount_: 0.008] := (1 - amount) * from
StepToFriend[from_, friends_, amount_: 0.06] :=
 from + amount * DirectionVector[from, from[[friends]]]
StepToEnemy[from_, enemies_, amount_: 0.02] :=
 from - amount * DirectionVector[from, from[[enemies]]]

(*Here we have the most vanilla dance fuction,
with step sizes and threshold given by
 the default settings.*)
Dance[x_, friends_, enemies_] := Module[{f = friends, e = enemies},
  {f, e} = ChooseNewPartner[f, e];
  {x // StepToCenter // StepToFriend[#, f] & // StepToEnemy[#, e] &, f, e}
 ]

(*This function gives all the parameter settings to play with later.*)
DanceParam[x_, friends_, enemies_, cs_, fs_, es_, thresh_] :=
 Module[{f = friends, e = enemies},
  {f, e} = ChooseNewPartner[f, e, thresh];
  {x // StepToCenter[#, cs] & // StepToFriend[#, friends, fs] & //
    StepToEnemy[#, enemies, es] &, f, e}
 ]

(*To make things easy, the code to plot the dance.*)
SeeDance[dancers_, plotrange_: 2] := Graphics[{PointSize[0.005], Opacity[0.6],
   Point[dancers]}, PlotRange → plotrange]
```

Here is the Vanilla Dance!

```
{x1, f1, e1} = InitializeDance[];
Dynamic[
 {x1, f1, e1} = Dance[x1, f1, e1];
 SeeDance[x1]
]
```

Now try Manipulating some of the parameters which determine the dance!

```
{x2, f2, e2} = InitializeDance[];
Dynamic@Manipulate[Graphics[{PointSize[0.005], Opacity[0.6], Dynamic[{x2, f2, e2} =
       DanceParam[x2, f2, e2, centersize, friendsize, enemysize, threshold];
      Point[x2]]}, PlotRange → plotrange], {{plotrange, 2, "Plot Range"}, 0.5, 5},
  {{centersize, 0.005, "Center Step Size"}, 0, 0.02},
  {{friendsize, 0.02, "Friend Step Size"}, 0, 0.1},
  {{enemysize, 0.01, "Enemy Step Size"}, 0, 0.1},
  {{threshold, 6, "Partner Switch Threshold"}, 0, 10}]
```

I've generated two pretty, different dances by changing these parameters

```
Module[{pts, friends, enemies},
 {pts, friends, enemies} = InitializeDance[];
 goodparams1 := {pts, friends, enemies, 0.0039, 0.0322, 0.01, 6};
 Graphics[{PointSize[0.007], Opacity[0.7],
   Dynamic[{pts, friends, enemies} = DanceParam @@ goodparams1;
    Point[pts]]}, PlotRange → 3]]
```

```
Module[{pts, friends, enemies},
 {pts, friends, enemies} = InitializeDance[];
 goodparams2 := {pts, friends, enemies, .00525, 0.0708, 0.01, 4.8};
 Graphics[{PointSize[0.007], Opacity[0.7],
   Dynamic[{pts, friends, enemies} = DanceParam @@ goodparams2;
    Point[pts]]}, PlotRange → 2]]
```

Dancing in Three Dimensions

Moving this dance to 3-dimensions is easy! Only two things needed to be done. The dancers needed to be initialized with a n×3 dimensional random array, and then Graphics needed to be changed to Graphic-

s3D. That's it!

```
SeeDance3D[dancers_, plotrange_:2] := Graphics3D[{PointSize[0.005], Opacity[0.6],
    Point[dancers]}, PlotRange → plotrange, SphericalRegion → True]
```

```
{x3, f3, e3} = InitializeDance[3];
Manipulate[
 Dynamic[
  {x3, f3, e3} =
   DanceParam[x3, f3, e3, centersize, friendsize, enemysize, threshold];
  SeeDance3D[x3, plotrange]], {{plotrange, 2, "Plot Range"}, 0.5, 5},
 {{centersize, 0.005, "Center Step Size"}, 0, 0.02},
 {{friendsize, 0.02, "Friend Step Size"}, 0, 0.1},
 {{enemysize, 0.01, "Enemy Step Size"}, 0, 0.1},
 {{threshold, 6, "Partner Switch Threshold"}, 0, 10}]
```

```
(*Do you know why this one FEELS faster?

   {x3,f3,e3}=InitializeDance[3];
Dynamic@Manipulate[Graphics3D[{PointSize[0.007],Opacity[0.6],Dynamic[
     {x3,f3,e3}=DanceParam[x3,f3,e3,centersize,friendsize,enemysize,threshold];
     Point[x3]]},PlotRange→plotrange,SphericalRegion→True],
  {{plotrange,2,"Plot Range"},0.5,5},
  {{centersize,0.005,"Center Step Size"},0,0.02},
  {{friendsize,0.02,"Friend Step Size"},0,0.1},
  {{enemysize,0.01,"Enemy Step Size"},0,0.1},
  {{threshold,6,"Partner Switch Threshold"},0,10}]*)
```

Multiple Partner Switches

In the vanilla dance by Woods, when it randomly comes time in the dance to switch partners, only one dancer chooses a new friend and enemy. I was interested in seeing the behaviour of a dance where a random number of dancers, m, switch their partners.

```
(*Random sampling of a Poisson distribution to
 determine number of dancers who choose new partners.
  μ determines the mean of this distribution,
keep this away from the number of total dancers.*)
m[μ_] := RandomVariate[PoissonDistribution[μ]]

(*As before, these function chooses new partners for random dancers.*)
ReChooseMulti[type_, is_] := Module[{m = Length[is], rules},
  rules = {#[[1]] -> #[[2]]} & /@ Transpose@{is, RandomInteger[{1, n}, m]};
  ReplacePart[type, Flatten@rules]]

ChooseNewPartner[friends_, enemies_, thresh_: 4, μ_: 1] := If[NewPartnerQ[thresh],
  i = RandomInteger[{1, n}, m[μ]];
  {ReChooseMulti[friends, i], ReChooseMulti[enemies, i]},
  {friends, enemies}]

(*I've updated the DanceParam function with this new parameter*)
DanceParamμ[x_, friends_, enemies_, cs_, fs_, es_, thresh_, μ_: 1] :=
 Module[{f = friends, e = enemies},
  {f, e} = ChooseNewPartner[f, e, thresh, μ];
  {x // StepToCenter[#, cs] & // StepToFriend[#, friends, fs] & //
    StepToEnemy[#, enemies, es] &, f, e}
 ]
```

Play around with the new parameter! Try to achieve a stable relationship between the threshold and number of dancers switching!

```
{x4, f4, e4} = InitializeDance[];
Dynamic@Manipulate[Graphics[{PointSize[0.007], Opacity[0.6], Dynamic[{x4, f4, e4} =
       DanceParamμ[x4, f4, e4, centersize, friendsize, enemysize, threshold, μ];
      Point[x4]]}, PlotRange → plotrange], {{plotrange, 2, "Plot Range"}, 0.5, 5},
  {{centersize, 0.005, "Center Step Size"}, 0, 0.02},
  {{friendsize, 0.02, "Friend Step Size"}, 0, 0.1},
  {{enemysize, 0.01, "Enemy Step Size"}, 0, 0.1},
  {{threshold, 6, "Partner Switch Threshold"}, 0, 10},
  {{μ, 1, "Mean of Dancers Switching"}, 1, 100}]
```

Distance Matrix

Here I was just trying to visualize the structure of the distance matrix given between dancers.

```
n = 1000;

(*Gives an m×m distance matrix between the first m dancers*)
DancerDistanceMatrix[dancers_, m_: 100] :=
 ArrayPlot[Table[Norm[dancers[[i]] - dancers[[j]]], {i, 1, m}, {j, 1, m}]]
(*Similar thing, but gives an upper triangle instead of a full matrix.*)
DancerDistanceTriangle[dancers_, m_: 100] :=
 ArrayPlot[Table[Norm[dancers[[i]] - dancers[[j]]], {i, 1, m}, {j, i + 1, m + 1}]]

{x5, f5, e5} = InitializeDance[];
Dynamic[{x5, f5, e5} = Dance[x5, f5, e5];
 {SeeDance[x5], DancerDistanceMatrix[x5, 100], DancerDistanceTriangle[x5, 100]}]

(*Just a note, I included both. Initially I was using the
 triangle because I figured it would be faster given that it
 isn't computing the distance between two points twice. However,
it turns out it's MUCH slower. If I was really interested I'm
 sure I could optimize this part much better.*)
{DancerDistanceMatrix[x5, 500] // Timing, DancerDistanceTriangle[x5, 499] // Timing}

{x6, f6, e6} = InitializeDance[];
Dynamic@
 Manipulate[{Graphics[{PointSize[0.005], Opacity[0.6], Dynamic[{x6, f6, e6} =
       DanceParamμ[x6, f6, e6, centersize, friendsize, enemysize, threshold, μ];
      Point[x6]]}, PlotRange → plotrange], Dynamic@DancerDistanceMatrix[x6, 100]},
  {{plotrange, 2}, 0.5, 5}, {{centersize, 0.005}, 0, 0.02},
  {{friendsize, 0.02}, 0, 0.1}, {{enemysize, 0.01}, 0, 0.1},
  {{threshold, 6}, 0, 10}, {μ, 1, 100}]
```

Clusters

My final experiment so far with the dancers is related to the density of the clusters they form. Woods' called this system a 'Boids swarm', and I found what this is. Boids, pronounced a thick accented New Yorker would 'birds', is an artificial life program given by Craig Reynolds in 1986. Reynolds' boids were given three simple instructions, and the emergent properties of this system is flocklike behaviour. Reyonld's three boid properties are:

1. separation - steer to avoid crowding nearby boids
2. alignment - steer in the average direction of nearby boids
3. cohesion - steer to move to the center of mass of nearby boids.

Now, I didn't want a flock of dancers, necessarily. However, I was interested in this notion of separation. I noticed that many of the dancers ended up in very dense clusters around each other, and I'd imagine that would be annoying at a dance. So here I include another step to the dance. At every beat of the dance, we assign each dancer to a cluster, given by *Mathematica*'s ClusteringComponents function. Then, we find the average position for each cluster. The new step to the dance is that every dancer now takes a step away from their personal cluster center.

```
(*Simple function to move away from given cluster centers*)
StepFromCluster[dancers_, centers_, amount_: 0.008] :=
 dancers - amount * DirectionVector[dancers, centers]

(*A Dance with steps away from cluster*)
DanceCluster[x_, friends_, enemies_] :=
 Module[{f = friends, e = enemies, components, clusters, clustermeans, centers},
  {f, e} = ChooseNewPartner[f, e];
  (*Determine the cluster each dancer belongs to*)
  components = ClusteringComponents[x, 5, 1];
  clusters = MapThread[List, {x, components}];
  (*Determine the average position of the cluster*)
  clustermeans = Mean /@ GroupBy[clusters, Last → First] // Values;
  centers = clustermeans[[components]];
  (*Include a step from those cluster centers*)
  {x // StepToCenter // StepToFriend[#, f] & // StepToEnemy[#, e] & //
    StepFromCluster[#, centers] &, f, e}
 ]

(*Same as above but with fine tuning of parameters*)
DanceClusterParam[x_, friends_,
  enemies_, cs_, fs_, es_, thresh_, μ_: 1, clus_: 0.02] :=
 Module[{f = friends, e = enemies, components, clusters, clustermeans, centers},
  {f, e} = ChooseNewPartner[f, e, thresh, μ];
  components = ClusteringComponents[x, 10, 1];
  clusters = MapThread[List, {x, components}];
  clustermeans = Mean /@ GroupBy[clusters, Last → First] // Values;
  centers = clustermeans[[components]];
  {x // StepToCenter[#, cs] & // StepToFriend[#, friends, fs] &//
     StepToEnemy[#, enemies, es] & // StepFromCluster[#, centers, clus] &, f, e}
 ]
```

See this less clustered dance in action

```
{x7, f7, e7} = InitializeDance[];
Dynamic[
 {x7, f7, e7} = DanceCluster[x7, f7, e7];
 SeeDance[x7]
]
```

Play with all the parameters!

```
n = 1000;
```

```
{x8, f8, e8} = InitializeDance[];
Manipulate[Dynamic[{x8, f8, e8} = DanceClusterParam[
     x8, f8, e8, centersize, friendsize, enemysize, threshold, μ, clus];
   SeeDance[x8, plotrange]], {{plotrange, 2, "Plot Range"}, 0.5, 5}, {{centersize,
    0.008, "Center Step Size"}, 0, 0.02}, {{friendsize, 0.06, "Friend Step Size"},
   0, 0.1}, {{enemysize, 0.02, "Enemy Step Size"}, 0, 0.1}, {{threshold, 5,
    "Partner Switch Threshold"}, 0, 10}, {{μ, 1, "Mean of Dancers Switching"},
   1, 100}, {{clus, 0.008, "Cluster Step Size"}, 0, 0.05}]
```

And of course, it all works in 3D! (also in n-dimensions but that's not so visual)

```
{x9, f9, e9} = InitializeDance[3];
Manipulate[Dynamic[{x9, f9, e9} = DanceClusterParam[
     x9, f9, e9, centersize, friendsize, enemysize, threshold, μ, clus];
   SeeDance3D[x9, plotrange]], {{plotrange, 2, "Plot Range"}, 0.5, 5}, {{centersize,
    0.008, "Center Step Size"}, 0, 0.02}, {{friendsize, 0.06, "Friend Step Size"},
   0, 0.1}, {{enemysize, 0.02, "Enemy Step Size"}, 0, 0.1}, {{threshold, 5,
    "Partner Switch Threshold"}, 0, 10}, {{μ, 1, "Mean of Dancers Switching"},
   1, 100}, {{clus, 0.008, "Cluster Step Size"}, 0, 0.05}]
```

Multiple Friends (Not Interesting!)