FLOW
(FLOcking Walkers)
Manual

Leonidas Eleftheriou and Ath. Kehagias
leonidas.eleftheriou@gmail.com and kehagiat@gmail.com

2013-06-06

1 Introduction

FLOW (**FLO**cking **W**alkers) is a Windows application which can be used for the generation and visualization of flocking and other types of self-organized collective motion. The main part of the application has been written in DarkBasic Professional and works on most current Windows versions.

FLOW is a simulator rather than a game. The user sets up certain parameters, hits the Go! button and then lets the walkers do their thing. The main inspiration for **FLOW** is Craig Reynolds' Boids work[1]. However, while Reynolds' model has *second order dynamics*, we use first order dynamics. These and other mathematical details are presented in Section 6. A word of caution is due here: we understand that including mathematical equations in a "game" manual is unusual. Our academic background shows through; however, we have tried to limit its manifestations in the last section of this manual. The user can use **FLOW** without reading any of the mathematics.

2 Installation Notes

The main part of **FLOW** is written in DarkBasic Professional. It is not necessary to have Dark Basic installed on your computer, but you must have DirectX 9.0c. (October 2006) or later. **FLOW** also contains a GUI interface which has been written in Visual Basic 6.0 and should also work on any Windows version. Both applications work on Windows XP, Windows 7 and Windows 8[2].

Installation is very simple. Unzip FLOW.zip[3] in a folder of your choice, for example in C:\FLOW. With Windows Explorer go to C:\FLOW and double click on the FLOW-GUI application. You are ready to go.

[1] For more information see the page http://www.red3d.com/cwr/boids/.

[2] These are the Windows versions we have tested; FLOW should work on other Windows versions as well.

[3] Which you download from http://users.auth.gr/kehagiat/Software.

3 Quick Start Guide

On double clicking the FLOW-GUI icon, the window of Figure 1 will appear.

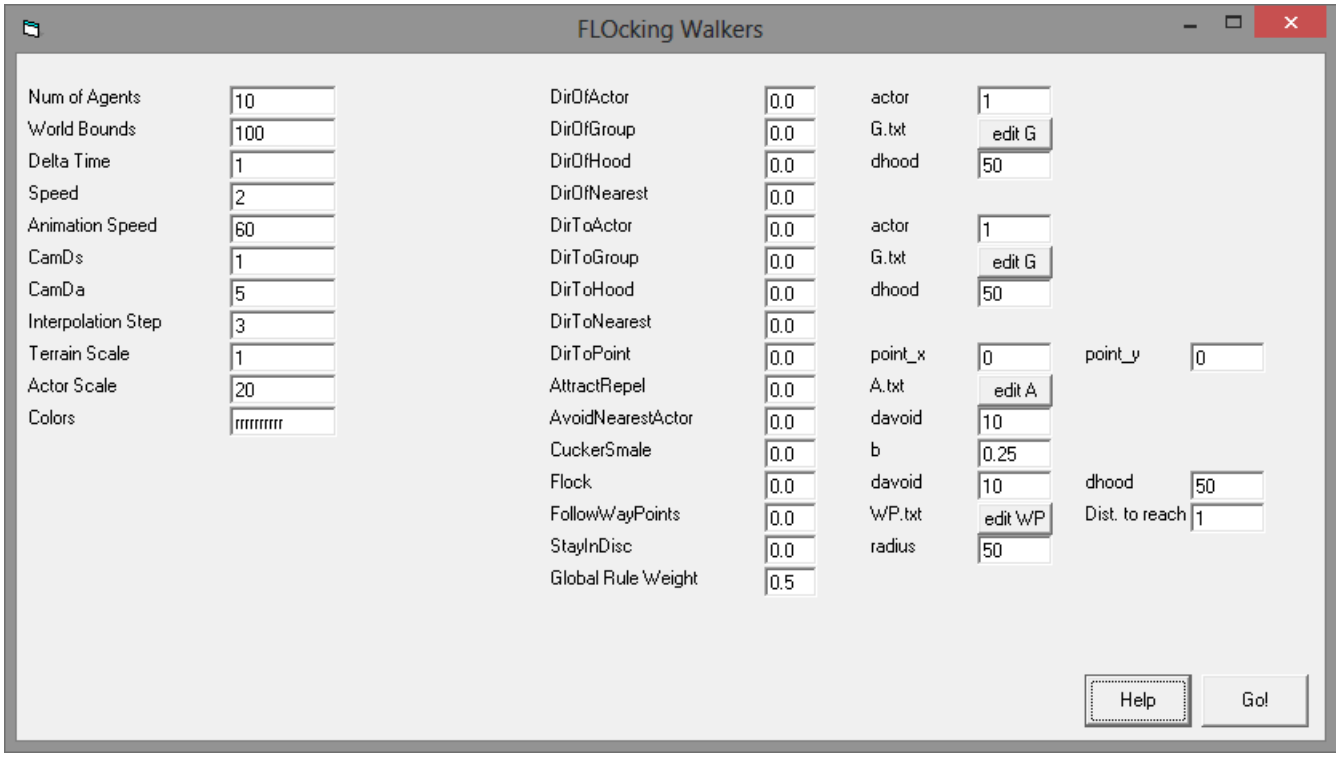

Figure 1: The FLOW GUI.

Go to the textbox labeled FollowWayPoints and change the value 0 to 1. Then hit the Go! button. After a few seconds the screen of Figure 2 will appear and the walkers will start walking.

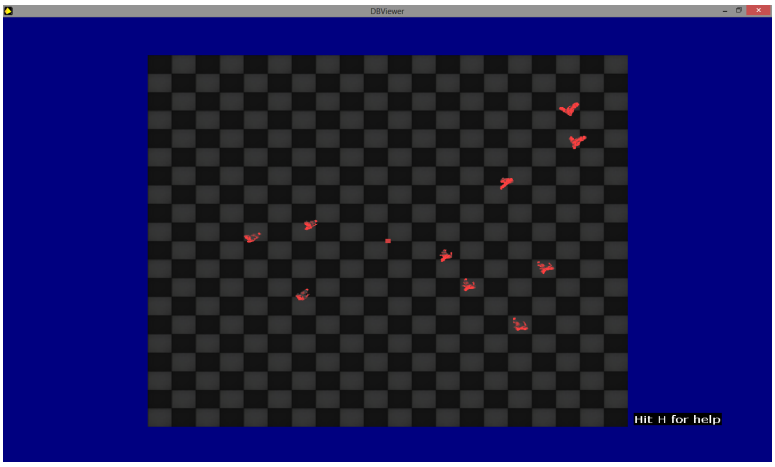

Figure 2: The beginning of the FollowWayPoints simulation.

In a short while the walkers will line up and will keep moving in a formation like the one illustrated in Figure 3, patrolling along the perimeter of a square.

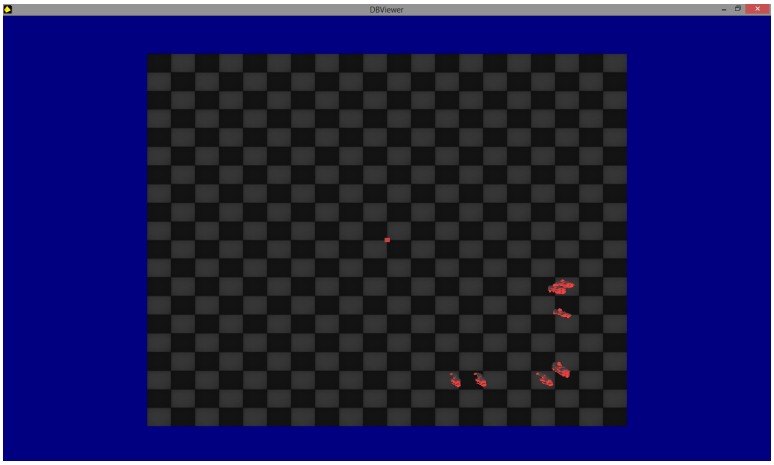

Figure 3: The evolution of the FollowWayPoints simulation.

You can look at the simulation from various points of view by typing any of the numbers `1-8`. The `Up / Down` arrow keys zoom the camera in / out; The `+ / -` keys move the camera up / down along the vertical axis. Type `h` to get a help screen; type `o` to go back to the simulation. To terminate the simulation, hit `Esc`.

The GUI can be divided into four areas, as seen in Figure 4.

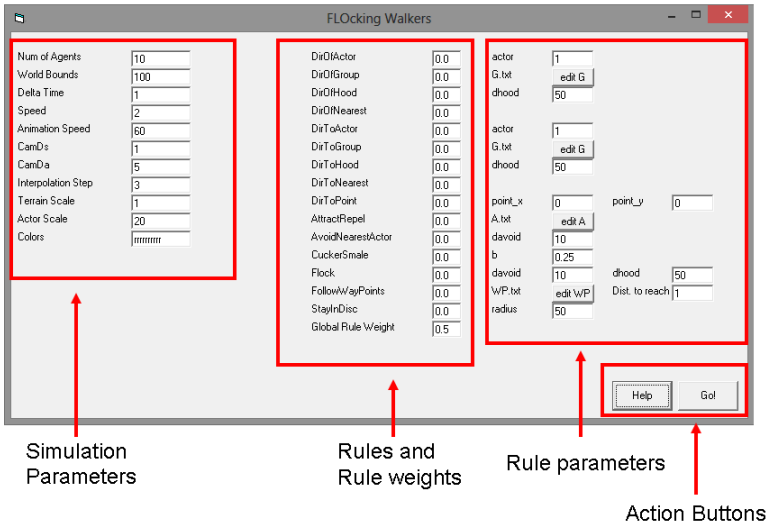

Figure 4: Organization of the FLOW GUI.

1. The first area allows one to set the parameters of the simulation. The names of the parameters are fairly self-explanatory. For example, Num of Agents controls the number of walkers. For a full explanation of all the parameters see Section 4 Manual.

2. The second area is the most important one. It contains the **rules** which determine the behavior of the walkers. The user types the **weight** of each rule in the textbox next to it; the rules are combined in proportion to their weights (when the weight of a rule is 0, the rule is inactive). We will soon give some examples of rule combinations. For a full explanation of each rule see Sections 4 and 6. **IMPORTANT:** the Global Rule Weight must be in the range 0 to 1 for the simulation to give correct results; when Global Rule Weight equals 0, the rules are inactive (even if their individual weights are nonzero).

3. The third area allows the user to set additional rule parameters. Once again, a detailed explanation appears in Sections 4 and 6.

4. The fourth area contains the *action buttons* Help and Go! which do the obvious things.

Let us give an example of setting up a simulation. Make sure you have exited the simulation window (if not, hit `Esc`); the FLOW-GUI window should still be active (if not, go back to C:\FLOW and double click on the FLOW-GUI application). Set the FollowWayPoints weight to 0, the DirOfHood weight to 0.5 and the AvoidNearestActor weight to 0.1; also set the dhood parameter to 50 and the davoid parameter to 10; finally, set the Global Rule Weight to 0.5. Now hit the Go! button at the bottom right corner.

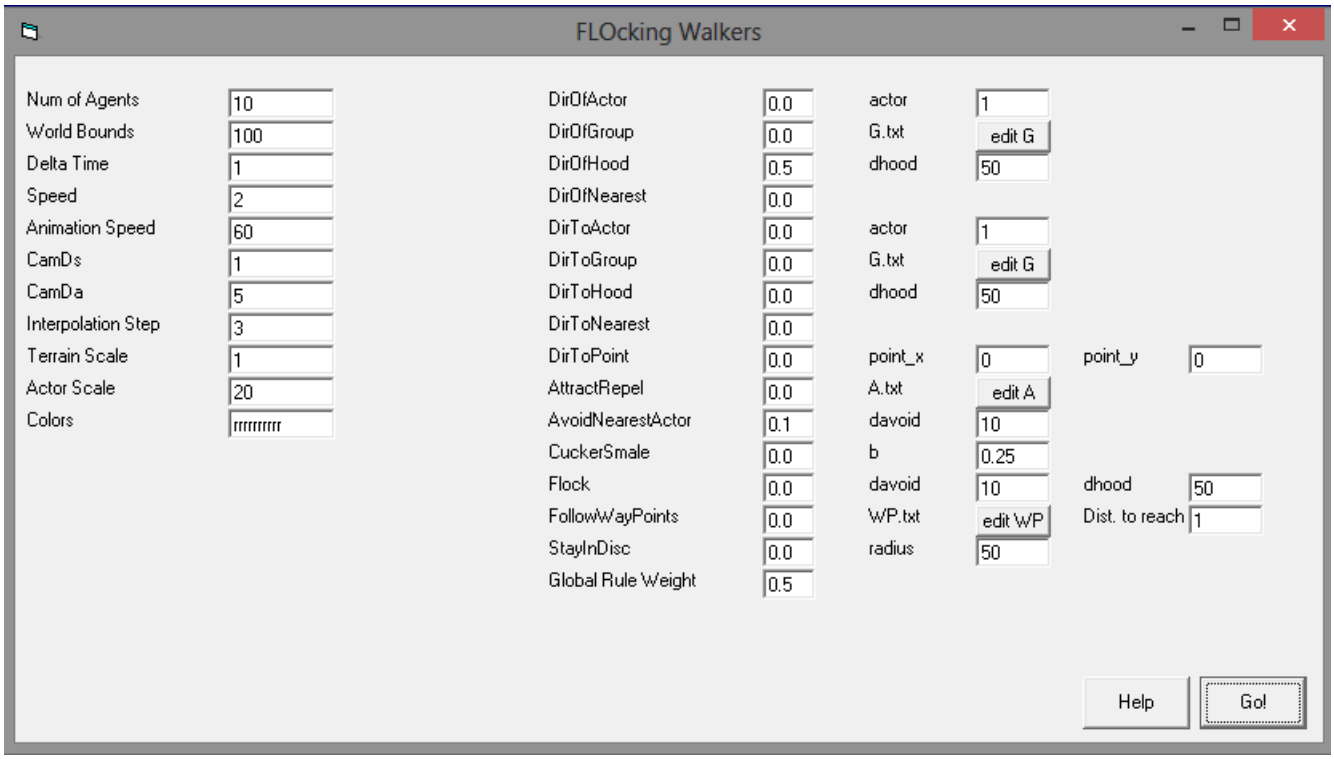

Figure 5: The GUI setup for the DirOfHood simulation.

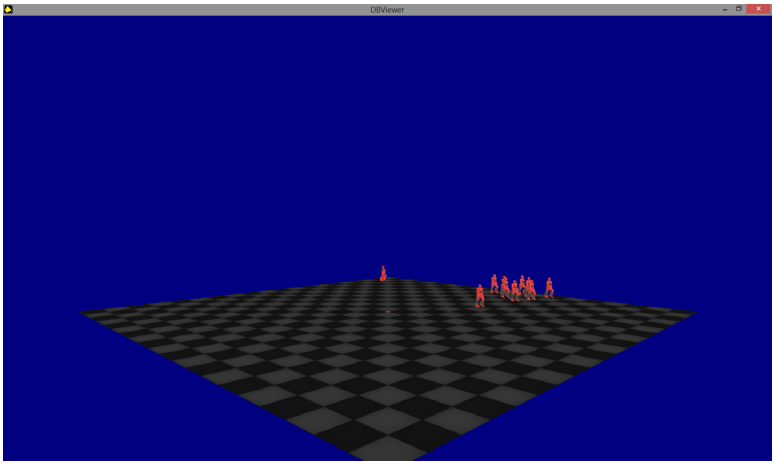

Figure 6: Boids-like behavior.

In this simulation the walkers have a "*Boids*-like" behavior, i.e., they tend to form up in a swarm, with all walkers having the same direction (the swarm direction). In Figure 6 you can see a snapshot in the middle stages of the simulation, when the swarm formation has set in.

Here is another example. Set up the the FLOW-GUI window to look exactly like in the Figure 7; in other words, set NumOfAgents to 5, AvoidNearestActor weight to 0.2, AttractRepel weight to 0.5 and the Global Rule Weight to 0.5. Now hit the Go! button at the bottom right corner.

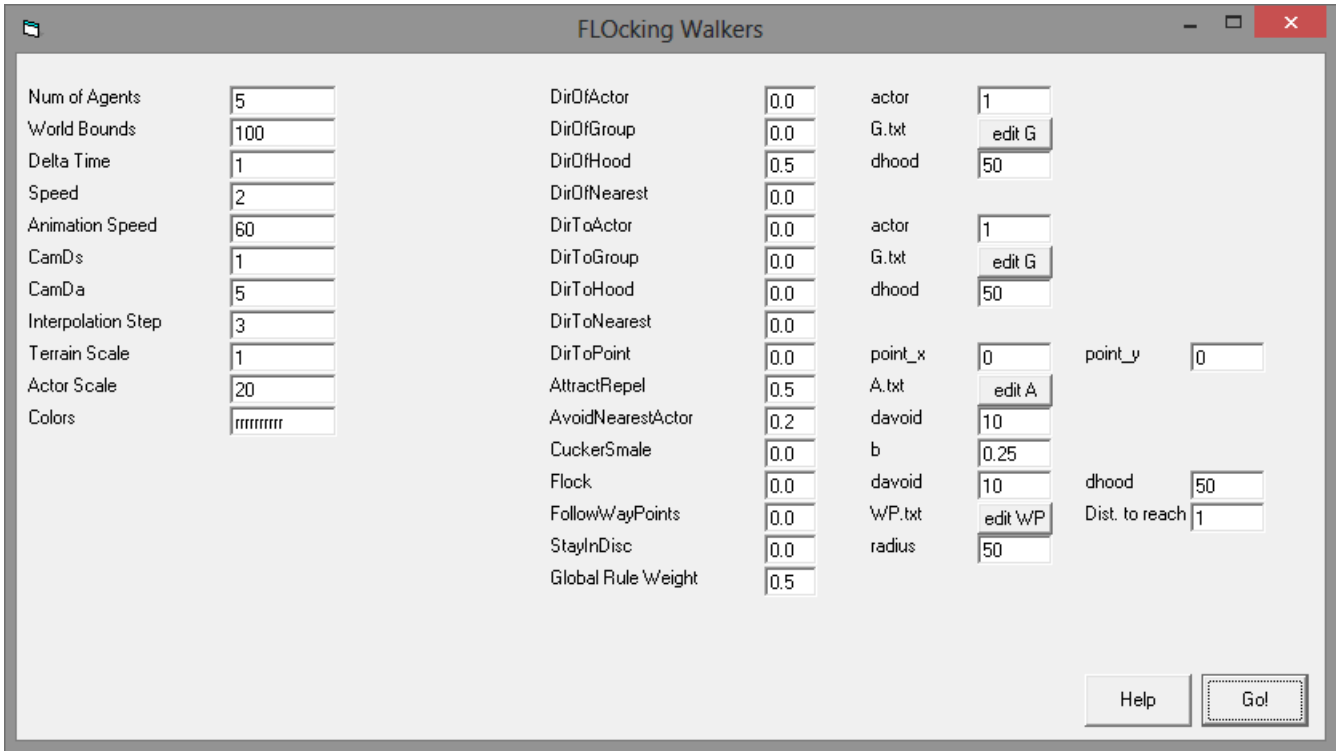

Figure 7: The GUI setup for the DirOfHood simulation.

When the steady state behavior sets in, one walker will perform a random walk and the remaining walkers will be following him. The situation looks like Figure 8.

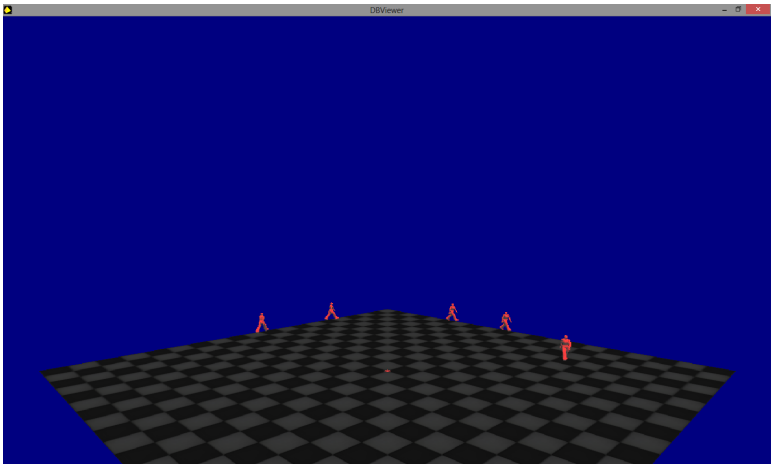

Figure 8: AttractRepel behavior.

4 Reference

4.1 Simulation Parameters

The following parameters control the simulation. In addition to the explanations given in the rest of this section, you can understand the significance of each parameter by trial and error. On the other hand, the default parameter values with which the GUI starts are "safe choices".

Num of Agents. Any integer N greater than zero, it is the total number of walkers.

World Bounds. Any number B greater than zero; the terrain in which the walkers move is a square with the four corners at $(-B, -B)$, $(-B, B)$, (B, B), $(B, -B)$.

Delta Time. A number dt greater than zero, equals the length of each simulation time step.

Speed. A number v greater or equal to zero, equals the speed of each walker. In one time step of the simulation, each walker covers a distance of $v \cdot dt$.

Animation Speed. A integer number $\overline{v}$ greater than zero; higher values make the walking animation play faster. But $\overline{v}$ does not influence the speed at which the walkers move through the terrain.

CamDs. A number d_s greater than zero, it is the linear speed at which the camera moves when the user presses the `Up / Down` arrow keys.

CamDa. A number d_a greater than zero, it is the angular speed at which the camera turns when the user presses the `Left / Right` arrow keys.

Interpolation Step. An integer number m greater than zero, it influences the speed at which the animation is played.

Terrain Scale. Leave this parameter equal to one.

Actor Scale. An number s greater than zero, it is used to increase / decrease the size of the walkers.

Colors. A string of characters which determines the color in which each walker is rendered. For example when the string is 'rgbrgbycg', the first walker is rendered in red, the second in green, the third in blue etc. If the length of string is greater than the number of walkers, additional characters are ignored; if it is smaller than the number of walkers, additional walkers are rendered in red.

4.2 Direction Change Rules

The walkers change their direction during the simulation according to the combined influence of several rules. Each of the rules is applied to every walker and produces a *direction*. Here we give a verbal description of the rules; an exact, mathematical description is given in Section 6.2.

It is important to understand how the *rule weights* work. First of all, we use a *global rule weight* c which must be between 0 and 1.

1. Small values of c make the combined influence of all rules small; the walker changes his direction slowly.

2. Large values of c increase the influence of the rules and the rate at which the walker changes his direction.

The weight of a rule determines its relative importance in the rule combination. What matters is the relative, not absolute value of a weight. For example, if rule 1 has weight $w_1 = 2$ and rule 2 has weight $w_2 = 4$, then rule 2 will be twice as important as rule 1; exactly the same would be true if $w_1 = 0.5$ and $w_2 = 1$.

In the following explanations the *current walker* is the one to which the rules are applied.

DirOfActor. This rules returns the direction of walker with number actor.

DirOfGroup. This rule returns the average direction of a group of walkers. Each walker can be associated with a different group; an N-by-N matrix G specifies the group of each walker as follows: if actor n belongs to the group of walker m, then $G_{m,n} = 1$, else $G_{m,n} = 0$. This matrix is contained in a file G.txt which can be edited by the user. To open G.txt in **Notepad**, hit the button edit G.

DirOfHood. This rule returns the average direction of the walkers who are in the *neighborhood* of the curent walker, i.e., all the walkers who have distance from the current walker less than or equal to dhood.

DirOfNearest. This rule returns the direction of the walker nearest to the current walker.

DirToActor. This rules returns the direction from the walker to the walker with number actor.

DirToGroup. This rules returns the direction from the current walker to the *barycenter* of a group of walkers. The group is specified by the matrix G described in the GroupDir rule.

DirToHood. This rule returns the direction from the current walker to the *barycenter* of the group of walkers contained in a neighborhood centered at the current walker and with radius dhood.

DirToNearest. This rules returns the direction from the current walker to the walker nearest to him.

DirToPoint. This rules returns the direction from the current walker to the point with coordinates point_x and point_y.

AttractRepel. This rule returns the direction which is the average of attractions and repulsions applied to the current walker from the other walkers. These attractions and repulsions are specified through an N-by-N matrix A:

1. if walker n attracts walker m, then $A_{m,n} = +$;
2. if walker n repels walker m, then $A_{m,n} = -$;
3. else $A_{m,n} = 0$.

The matrix A is contained in a file A.txt which can be edited by the user. To open A.txt in **Notepad**, hit the button edit A.

AvoidNearestActor. This rules returns the direction which points away from the walker who is closest to the current walker.

CuckerSmale. This rules returns the direction obtained from the Cucker-Smale flocking model.

Flock. This rules returns a direction which results from flocking dynamics; basically it is a combination of DirOfHood and AvoidNearestActor.

FollowWayPoints. This rule returns a direction varying with the position of the current walker, in such a manner that the walker visits a sequence of M waypoints. The waypoints are listed in the M-by-2 matrix W, with the x and y coordinates of the m-th waypoint contained in the m-th row of the matrix. W is contained in a file WP.txt which can be edited by the user. To open WP.txt in **Notepad**, hit the button edit WP.

StayInDisc. This rules returns a direction which is varying with the position of the walker so as to always keep him inside a disc with center at $(0, 0)$ and radius radius.

Global Rule Weight. This is not a rule, but a global weight which is applied to all rules. It is the number c described in the beginning of this section.

5 Some Programming Notes

As already mentioned the FLOW-GUI has been programed in Visual Basic 6 and the Simulator DBV2 in Dark Basic Professional. The code for these applications is contained in the subfolder FLOW\Code. Feel free to modify it as you wish. The Visual Basic project file is FLOW-GUI.vbp; the Dark Basic project file is FLOW Viewer.dbpro.

The FLOW-GUI has a single purpose: to make it easier to enter the simulation parameters, the rule weights etc. When you hit the Go! button, FLOW-GUI writes this information into two files: FLOW Input.txt and FLOW Rules.txt which are then used by FLOW Viewer. If you prefer, you can edit these files with a text editor (they are contained in the FLOW folder, their syntax shoul dbe obvious) and then click on the FLOW Viewer application to get the simulation (and the visuals) started.

6 Some Mathematics Notes

6.1 The Mathematical Model

The mathematical model behind the application consists of equations describing the movement of N walkers. Each walker has *position*, *direction* and *speed*.

1. The speed v is fixed throughout the simulation and the same for all walkers.

2. The position of the n-th walker at time t is given by

$$\mathbf{r}_n(t) = [x_n(t), y_n(t)]$$

   where $x_n(t)$ is the x coordinate of the n-th walker at time t and $y_n(t)$ is the y coordinate of the n-th walker at time t.

3. The direction of the n-th walker at time t is given by the *direction vector*

$$\mathbf{u}_n(t) = [p_n(t), q_n(t)],$$

   where $p_n(t)$ is the x coordinate of the vector and $q_n(t)$ is the y coordinate of the vector (both of these for the n-th walker at time t).

These variables are illustrated in Figure 9.

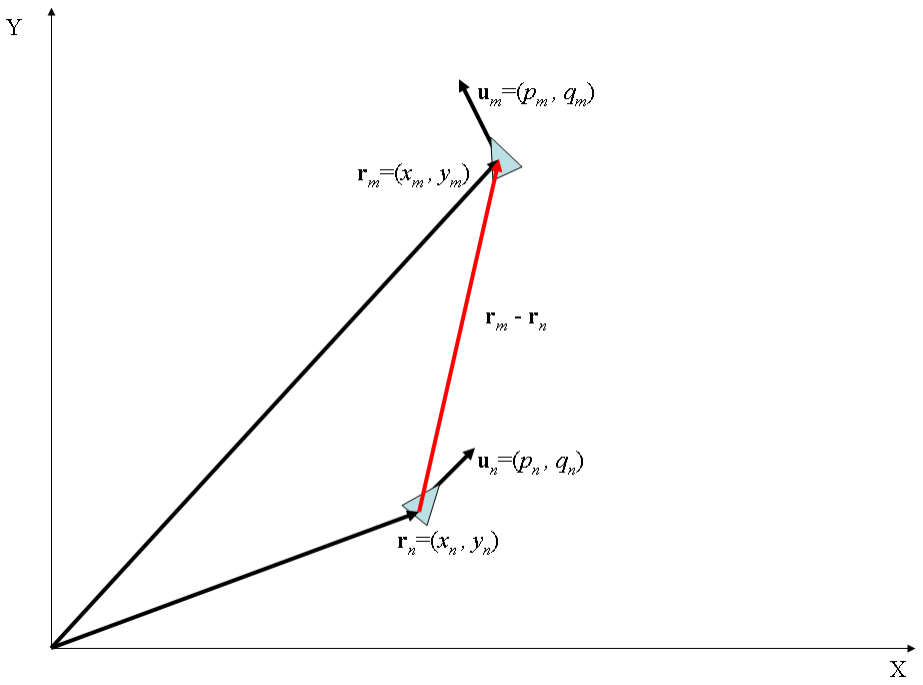

Figure 9: The walkers in the plane.

So we work with two-dimensional, time varying vectors such as $\mathbf{r}_n(t)$ and $\mathbf{u}_n(t)$. It is useful at this point to introduce the *length* of a vector: if $\mathbf{w} = [a, b]$, then the length of $\mathbf{w}$ is $\|\mathbf{w}\| = \sqrt{a^2 + b^2}$.

The time variable t takes integer values $0, 1, 2, \ldots$. But the user also has access to a time step dt, so that the simulation evolves in discrete time steps $0 \cdot dt$, $1 \cdot dt$, $2 \cdot dt$, ...

The position of the n-th walker evolves according to the following equation

$$\mathbf{r}_n(t+1) = \mathbf{r}_n(t) + v \cdot \mathbf{u}_n(t+1) \cdot dt.$$

So the factor that determines the trajectory of the n-th walker is the direction $\mathbf{u}_n(t)$ and what creates interesting trajectories is the way in which $\mathbf{u}_n(t)$ changes in time. This is determined by a combination of **rules**. More specifically, the direction at time $t+1$ is determined by the following equation

$$\mathbf{u}_n(t+1) = \frac{1}{W}\left[(1-c) \cdot \mathbf{u}_n(t) + c \cdot \mathbf{\Delta u}_n(t+1)\right] \tag{1}$$

where c is a parameter taking values between 0 and 1. If c is small, then $\mathbf{u}_n(t+1)$ is pretty close to $\mathbf{u}_n(t)$; if c is large, then $\mathbf{u}_n(t+1)$ can become quite different from $\mathbf{u}_n(t)$, depending on the **total direction update** $\mathbf{\Delta u}_n(t+1)$. In turn, the total direction update is the sum of the direction updates effected by the **active rules:**

$$\mathbf{\Delta u}_n(t+1) = w_1 \mathbf{\Delta u}_n^1(t+1) + w_2 \mathbf{\Delta u}_n^2(t+1) + \ldots$$

Each direction update $\mathbf{\Delta u}_n^k(t+1)$ depends on the positions and directions of all the walkers

$$\mathbf{\Delta u}_n^k(t+1) = F\left(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t), \mathbf{u}_1(t), \mathbf{u}_2(t), \ldots, \mathbf{u}_N(t)\right).$$

The nonnegative number w_k is the weight of the k-th rule. If $w_k = 0$, the k-th rule is inactive. The absolute values of the w_k's are not important; but *relatively* larger values of w_k cause a stronger effect of the k-th rule on $\mathbf{\Delta u}_n(t+1)$. In other words, if $w_1 = 1$ and $w_2 = 2$, the second rule is twice as strong as the first one; and exactly the same is true when $w_1 = 0.1$ and $w_2 = 0.2$.

Finally, in (1) the number $W = \|(1-c)\cdot \mathbf{u}_n(t) + c\cdot \mathbf{\Delta u}_n(t+1)\|$, the length of the vector $(1-c)\cdot \mathbf{u}_n(t) + c\cdot \mathbf{\Delta u}_n(t+1)$. So we can rewrite (1) as follows

$$\mathbf{u}_n(t+1) = \frac{(1-c)\cdot \mathbf{u}_n(t) + c\cdot \mathbf{\Delta u}_n(t+1)}{\|(1-c)\cdot \mathbf{u}_n(t) + c\cdot \mathbf{\Delta u}_n(t+1)\|}. \tag{2}$$

In other words, the direction vector always has length one: $\|\mathbf{u}_n(t+1)\| = 1$. This is enforced so that the distance traveled by every walker at every time step is $v \cdot dt$.

What remains to be explained is the form of each of the available rules. This wil be done in the next section.

6.2 Rules

In what follows we will describe precisely each one of our rules. The following things should be kept in mind.

1. Each rule is applied at every time step of the simulation and to every walker, i.e., for $n = 1, 2, .., N$ (where N is the total number of walkers).
2. The walker to whom the rule is applied will be called the *current walker* and we will assume that his number is n.

DirOfActor. The direction output by this rule is

$$\Delta\mathbf{u} = \mathbf{u}_m(t)$$

where m is the actor number (input provided to the rule).

DirOfGroup . The direction output by this rule is

$$\Delta\mathbf{u} = \frac{\sum_{i=1}^{N} G_{n,i}\cdot \mathbf{u}_i(t)}{\left|\left|\sum_{i=1}^{N} G_{n,i}\cdot \mathbf{u}_i(t)\right|\right|}$$

where the $N \times N$ matrix G (contains 0's and 1's) is the input provided to the rule.

DirOfHood. The direction output by this rule is

$$\Delta\mathbf{u} = \frac{\sum_{i\in V} \mathbf{u}_i(t)]}{\left|\left|\sum_{i\in V} \mathbf{u}_i(t)]\right|\right|}$$

where V is the set of walkers contained in the neighborhood which is centered at the n-th (current) walker and has radius d_{hood} (where d_{hood} is the input provided to the rule). In other words

$$V = \{m : \|\mathbf{r}_m(t) - \mathbf{r}_n(t)\| \le d_{hood}\}.$$

DirOfNearest. The direction output by this rule is

$$\Delta \mathbf{u} = \mathbf{u}_k(t)$$

where

$$k = \arg\min_{i \neq n} ||\mathbf{r}_i(t) - \mathbf{r}_n(t)|| .$$

DirToActor. The direction output by this rule is

$$\Delta \mathbf{u} = \frac{\mathbf{r}_m(t) - \mathbf{r}_n(t)}{||\mathbf{r}_m(t) - \mathbf{r}_n(t)||}$$

where m is the actor number (input provided to the rule).

DirToGroup. The direction output by this rule is

$$\Delta \mathbf{u} = \frac{\frac{\sum_{i \in V} G_{n,i} \mathbf{r}_i(t)}{\sum_{i \in V} G_{n,i}} - \mathbf{r}_n(t)}{\left|\left| \frac{\sum_{i \in V} G_{n,i} \mathbf{r}_i(t)}{\sum_{i \in V} G_{n,i}} - \mathbf{r}_n(t) \right|\right|}$$

where the $N \times N$ matrix G (contains 0's and 1's) is the input provided to the rule.

DirToHood. The direction output by this rule is

$$\Delta \mathbf{u} = \frac{\left(\frac{1}{M} \sum_{i \in V} \mathbf{r}_i(t)\right) - \mathbf{r}_n(t)}{\left|\left| \left(\frac{1}{M} \sum_{i \in V} \mathbf{r}_i(t)\right) - \mathbf{r}_n(t) \right|\right|}$$

where V is the set of walkers contained in the neighborhood which is centered at the n-th (current) walker and has radius d_{hood} (where d_{hood} is the input provided to the rule). In other words

$$V = \{m : \|\mathbf{r}_m(t) - \mathbf{r}_n(t)\| \leq d_{hood}\} .$$

M is the number of walkers contained in V.

DirToNearest. The direction output by this rule is

$$\Delta \mathbf{u} = \frac{\mathbf{r}_k(t) - \mathbf{r}_n(t)}{||\mathbf{r}_k(t) - \mathbf{r}_n(t)||}$$

where

$$k = \arg\min_{i \neq n} ||\mathbf{r}_i(t) - \mathbf{r}_n(t)|| .$$

DirToPoint. The direction output by this rule is

$$\Delta \mathbf{u} = \frac{\mathbf{r_0} - \mathbf{r}_n(t)}{||\mathbf{r_0} - \mathbf{r}_n(t)||}$$

where $\mathbf{r_0} = [x_{point}, y_{point}]$ is the input provided to the rule.

AttractRepel. The direction output by this rule is

$$\Delta\mathbf{u} = \frac{\sum_{i=1}^{N} A_{n,i} \cdot (\mathbf{r}_i(t) - \mathbf{r}_n(t))}{\left|\left|\sum_{i=1}^{N} A_{n,i} \cdot (\mathbf{r}_i(t) - \mathbf{r}_n(t))\right|\right|}$$

where the $N \times N$ matrix A (contains 0's, 1's and -1's) is the input provided to the rule.

AvoidNearestActor. The direction output by this rule is

$$\Delta\mathbf{u} = \begin{cases} \frac{\mathbf{r}_k(t) - \mathbf{r}_n(t)}{\|\mathbf{r}_k(t) - \mathbf{r}_n(t)\|} & \text{if } ||\mathbf{r}_k(t) - \mathbf{r}_n(t)|| \leq d_{avoid} \\ [0, 0] & \text{if } ||\mathbf{r}_k(t) - \mathbf{r}_n(t)|| > d_{avoid} \end{cases}$$

where

$$k = \arg\min_{i \neq n} ||\mathbf{r}_i(t) - \mathbf{r}_n(t)||$$

and d_{avoid} is the input provided to the rule.

FollowWayPoints. The input to this rule is an M-by-2 matrix W which contains in each row the coordinates of a *waypoint*. The rule keeps outputting a direction $\Delta\mathbf{u}$ which points to the first waypoint until the current walker gets to withing d (input to the rule) of this waypoint; then the rule outputs a direction $\Delta\mathbf{u}$ which points to the second waypoint until the current walker gets to within d of this waypoint and so on until the waypoints are exhausted; then the rule goes back to the first waypoint and continues like this in a cyclic order.

StayInDisc. The direction output by this rule is

$$\Delta\mathbf{u} = \begin{cases} [0, 0] & \text{if } \mathbf{r}_n(t) \leq d_{radius} \\ \frac{-\mathbf{r}_n(t)}{||\mathbf{r}_n(t)||} & \text{if } \mathbf{r}_n(t) > d_{radius} \end{cases}$$

where d_{radius} is the input provided to the rule.